University of Alberta

C•RAM WITH A FAULT-TOLERANT RECONFIGURABLE 1D,2D AND 3D COMMUNICATION NETWORK

by

Daniel Arie Leder ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2004

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-96507-4
Our file Notre référence
ISBN: 0-612-96507-4

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canadä

Abstract

This thesis introduces a reconfigurable interconnection network for Computational RAM (C•RAM) that incorporates 1D, 2D and 3D communication between processing elements. The processing element mesh is designed to be a dynamically reconfigurable interconnection network with fault-tolerance extending into all three dimensions of the mesh. A novel memory bank architecture for C•RAM is also proposed that allows for computations and load/store operations to occur simultaneously.

The efficiency of operation for C•RAM is enhanced by the introduction of the dynamically reconfigurable interconnection network, since shifting can be performed more efficiently. The efficiency of C•RAM is also enhanced by the memory bank architecture, since computations are maximized by allowing computations to be performed in parallel with external memory access.

Acknowledgements

This research was supported by Micronet R&D, MOSAID Technologies Inc, and the University of Alberta. I would like to give special thanks to my supervisor, Duncan Elliott for the help and direction. I would also like to thank Tyler Brandon, Craig Joly and John Koob for listening, and letting me bounce ideas off of them. I would like to thank Kelsey Thompson for her efforts in reading and editing my thesis.

Table of Contents

1 Introduction **1**
  1.1 Thesis Organization . . . 3

2 Prior Art **4**
  2.1 Computational RAM (C•RAM) Architecture . . . 4
    2.1.1 Overview . . . 4
    2.1.2 Generic C•RAM Architecture . . . 5
    2.1.3 3DSOI C•RAM . . . 8
  2.2 Silicon-on-Insulator . . . 11
    2.2.1 Overview . . . 11
    2.2.2 MIT Lincoln Labs - 3D SOI . . . 12
  2.3 Grid Based Processors . . . 14
    2.3.1 Network Topologies . . . 14
    2.3.2 Implemented Grid Based Processors . . . 16
  2.4 Redundancy . . . 19
    2.4.1 Logic Redundancy . . . 20
    2.4.2 Memory Redundancy . . . 20
    2.4.3 C•RAM Redundancy . . . 22
    2.4.4 Grid Based Redundancy . . . 22
    2.4.5 Altera's 2D Programmable Redundancy . . . 24
  2.5 Summary . . . 24

3 Architectural Concepts **26**
  3.1 Memory Array . . . 26
    3.1.1 Multi-bank Design . . . 28
  3.2 Interconnection Network . . . 31
    3.2.1 Applications of a Multi-dimensional Network . . . 32
    3.2.2 Architecture of a Multi-dimensional Network . . . 33
    3.2.3 Architecture of the 3D Network . . . 34
    3.2.4 A 1D Network in a 3D Network . . . 35
    3.2.5 A 2D Network in a 3D Network . . . 36
    3.2.6 Multiple Chip Interconnection . . . 38
  3.3 Redundancy Design . . . 41
    3.3.1 Compatibility with Memory Redundancy . . . 41

3.3.2 Processing Element Redundancy 46
3.3.3 Interconnection Redundancy 53
3.4 Summary 55

4 Implementation **56**
4.1 MIT Lincoln Labs 0.18μm SOI 56
4.2 Memory Array 57
4.2.1 Multi-bank Memory 57
4.2.2 Multi-bank Memory Controller 67
4.2.3 Memory Redundancy 71
4.3 Processing Element 73
4.4 Processing Element Redundancy 75
4.4.1 Row & Column Redundancy 75
4.4.2 Redundancy Controller 77
4.5 Interconnection Network 78
4.5.1 Broadcast Bus Interconnection 79
4.5.2 1D, 2D, and 3D Interconnection 80
4.5.3 Interconnection Network Controller 92
4.5.4 Off-chip Interconnection Controller 92
4.6 3D SOI Stack Implications 93
4.7 Performance 94
4.7.1 Design Performance 95
4.8 Design Results 98
4.8.1 Power - Estimation 99
4.8.2 Area - Estimation 102
4.8.3 Yield 107
4.9 Summary 112

5 Testing **114**
5.1 Testing of First Generation 3DSOI C•RAM 114
5.1.1 Proof of 3DSOI Technology 115
5.1.2 Test Strategy 117
5.1.3 Test Results 119
5.2 Summary 124

6 Conclusions **125**
6.1 Future Work 127

Bibliography **128**

A Yield Calculations **130**
A.1 Column and PE Bus Redundancy 130
A.2 Dependent Column and PE Redundancy 131
A.3 Independent Column and PE Redundancy 132

List of Tables

2.1 PDSOI versus FDSOI [11] 12

3.1 Plane Addressing Calculation on Failure 53

4.1 Sense Amplifier Indexing 68
4.2 Multi-Bank Controller Operation 69
4.3 Circuit Propagation Delay for 1.2V & 1.5V Operation 97
4.4 Maximum Operational Frequencies for 1.2V & 1.5V 98
4.5 Dynamic Power Calculations for 1.5V Operation 100
4.6 Dynamic Power Calculations for 1.2V Operation 101
4.7 Dynamic Power for Select Circuits 102
4.8 Height Approximations of Shift and Redundancy Wiring 104
4.9 Length Approximations of Shift and Redundancy Wiring (single die) 104
4.10 Area Approximations for 4x8x4 Array with Redundancy 105
4.11 Area Approximations for 4x8x4 Array Redundancy 106

5.1 3DSOI C•RAM Test Results 121

List of Figures

2.1 C•RAM Array - Memory cell array incorporating PEs [8] 6
2.2 C•RAM PE - 256-Function PE including interconnection [8] 7
2.3 3DSOI C•RAM Organization [16] 9
2.4 3DSOI C•RAM Processing Element [16] 10
2.5 MIT Lincoln Labs 3D SOI Stack 13
2.6 Network Topologies [8, 2] 15
2.7 Conventional Column Redundancy Architecture [9] 21
2.8 Shift Column Redundancy Architecture [9] 22
2.9 Grid Based Redundancy Approaches [17] 23

3.1 Multi-Bank C•RAM 30
3.2 C•RAM - One-dimensional Interconnect in Three-dimensional Mesh 36
3.3 C•RAM - Two-dimensional Interconnect in Three-dimensional Mesh 37
3.4 Off-chip Data Shifting 39
3.5 Multi-chip 1D and 3D Problem 40
3.6 C•RAM Redundancy of Processing Elements and Memory (original) 42
3.7 C•RAM Redundancy of Processing Elements and Memory (option 1) 43
3.8 C•RAM Redundancy of Processing Elements and Memory (option 2) 44
3.9 Yield Plots for Memory and PE redundancy 45
3.10 C•RAM Grid Redundancy using Redundant Bus 47
3.11 C•RAM Redundancy of Processing Elements 48
3.12 Two-dimensional PE Redundancy 50
3.13 90% Yield for PE Redundancy 52
3.14 C•RAM Interconnection Redundancy 54

4.1 Multi-Bank Architecture with 4 Banks per PE 58
4.2 Word-line to SRAM cell 60
4.3 Word-line Decoding (option1) 60
4.4 Word-line Decoding (option2) 60
4.5 Sense Amplifier (M Register) 62
4.6 Write-Back M Register to Cell 63
4.7 Sense Amplifier, Precharge & Column Controller 63
4.8 Column Decoding 64
4.9 Redundancy of Column Decoding 65
4.10 Address Recalculation for Address Continuity 66

4.11 Plane Address Recalculation for Address Continuity 67
4.12 Multi-bank Memory Controller 71
4.13 Memory and PE Replacement 72
4.14 Memory Redundancy Schematic Detail 73
4.15 Processing Element 74
4.16 Processing Element Redundancy 76
4.17 Two-dimensional PE Redundancy 76
4.18 PE Grid Redundancy Schematic 77
4.19 PE Grid Redundancy Detail 78
4.20 Shift Circuit 82
4.21 Shift Circuit with z Redundancy 82
4.22 Shift Circuit Interconnection 84
4.23 Interconnections for 1D and 3D using Spiral 85
4.24 Interconnections for 2D using Spiral 85
4.25 PE Interconnections with a Failed Plane 86
4.26 Spiral Interconnection Circuit with Redundancy 87
4.27 Torus with Similar Delay Links 88
4.28 Torus Across Multiple Arrays 88
4.29 Array Block Interconnection over Multiple Array Blocks 90
4.30 3D via (a)symbol, and (b) stacking 94
4.31 Possible PE placement for 2D Grid 102
4.32 Area Bar Graph 106
4.33 Yield Plots for PE redundancy 108
4.34 Equivalent Yield for PE Array 109
4.35 Equivalent Yield for 64x64 Fault-tolerant Grid 110
4.36 Equivalent Yield for 64x64x8 Fault-tolerant Array 110
4.37 Yield Plots for Memory Redundancy 111
4.38 Equivalent Yield for Memory Redundancy 112

5.1 Chip Plots - a black dot represents a good cell 122
5.2 Chip Plots 122
5.3 Contour Shmoo Plot of Good Cells of Chip 1 123
5.4 Contour Shmoo Plot of Errors of Chip1 124

List of Symbols & Abbreviations

ALU	Arithmetic Logic Unit
C•RAM	Computational Random Access Memory
CVD	Chemical Vapour Deposition
DRAM	Dynamic Random Access Memory
FDSOI	Fully-Depleted Silicon-On-Insulator
I/O	Input-Output
IC	Integrated Circuit
MUX	Multiplexer
NEWS	North-East-West-South (interprocessor communication)
PDSOI	Partially-Depleted Silicon-On-Insulator
PE	Processing Element
PIM	Processor-in-Memory
RAM	Random Access Memory
SIMD	Single Instruction stream Multiple Data streams
SRAM	Static Random Access Memory
SOI	Silicon-On-Insulator

Chapter 1

Introduction

This thesis presents architectural enhancements and circuit designs for Computational RAM (C•RAM) for implementation in a three-dimensional (3D) integrated circuit (IC) process. C•RAM is a Single Instruction stream Multiple Data stream (SIMD) processor-in-memory parallel processing architecture. Enhancing the interprocessor communication capabilities while maintaining design reliability is presented. This architecture involves the addition of processing elements to conventional memory, where each processing element is built in the pitch of a small number (1-4) of memory columns [8]. This thesis will contribute to the creation of a dynamically reconfigurable interconnection network that will allow for one-dimensional, two-dimensional, and three-dimensional communication between processing elements. How the addition of redundancy to the design accommodates all dimensions of communication will also be shown. The main contribution will be transforming a linear array of processing elements (PEs) into a fault-tolerant three-dimensional cube of PEs. The target IC process is the Massachusetts Institute of Technology Lincoln Labs (MIT LL) developed fully-depleted silicon-on-insulator (FDSOI) process where the wafers can be stacked on top of each other, thereby automatically adding a third dimension to chip creation. During previous work, techniques to take advantage of this process were discovered that allow for a single design to be used and stacked multiple times to create a scalable architecture. The designer is only required to determine the maximum height, since it would be necessary to be able to address all dies in the stack. This would require the designer to anticipate the number of dies needed to accomplish the design.

This has led to the architecture changes that are discussed in this thesis, where the stacking of dies is used to reduce the number of interconnections on a single wafer to create a complete $N \times N \times N$ cube. With stacking capabilities it is possible to make each die a $N \times N$ plane of the cube, thereby reducing the number of interconnection wires per die by N. Since the current stack height has been limited to two wafers, each die design will be based on a $N/2 \times N \times N$ cube, which leads to a single order of magnitude decrease in the number of interconnection wires per die. Due to the fact that a new dimension has been added to chip design, design techniques for dividing a design and maintaining single die functionality are needed. These techniques allow for additional scaling of the stack while ensuring correct functionality that meets the design specifications. Some techniques for preserving functionality of single to multiple die stacks will be researched and discussed in this thesis.

An additional contribution of this thesis will be enhancing the memory interface of the C•RAM architecture. Previous architectures have been limited to loading or storing the memory of the C•RAM before computations could be performed. The limitation has been that during loading or storing of the memory, computations cannot be performed, thereby reducing the efficiency of the C•RAM architecture. The memory interface of the C•RAM architecture is enhanced by implementing multiple banks of memory per processing element, allowing for external memory accesses to be performed in parallel with the processing element computations. In this way, the efficiency of the C•RAM architecture is improved so that the processing elements are operating as much as possible.

A prior generation C•RAM architecture manufactured in the MIT LL process is tested to determine the viability of the MIT LL process and to determine the amount of redundancy that may be needed to improve the yield of the design being done in the MIT LL process.

1.1 Thesis Organization

In this thesis, Chapter 2 presents an overview of past work in fields of SIMD architectures, silicon-on-insulator (SOI) technology, processor interconnection networks and redundancy. In Chapter 3, design trade-offs and architecture enhancements of C•RAM are discussed. To show that the architecture enhancements are viable, Chapter 4 describes the design of the new architecture. Chapter 5 presents the test results of the prototype chip of the prior SOI C•RAM architecture. Chapter 6 summarizes and presents the conclusions for the thesis.

Chapter 2

Prior Art

This chapter presents four sections: Computational RAM, 3D SOI, grid-based processors, and redundancy for memory and logic.

Section 2.1, Computational RAM, introduces the developed C•RAM architectures. The generic C•RAM architecture, a commercial and a research design, are discussed to show the viability of the design.

Section 2.2, 3D SOI, introduces MIT Lincoln Labs' silicon-on-insulator (SOI) technology that allows for the stacking of multiple SOI wafers (three-dimensional).

Section 2.3, Grid Based Processors, takes a look at past implementations of grid-based processors and the processor interconnection methods used.

Section 2.4, Redundancy, looks at prior methods of redundancy used for memories, and grid-based logic.

2.1 Computational RAM (C•RAM) Architecture

2.1.1 Overview

Computational RAM (C•RAM) is a processors-in-memory (PIM) parallel processing architecture, sometimes known as a logic-enhanced memory. The main concept in C•RAM is to implement computational units near the memory core of a chip. This concept allows the designer to take full advantage of the bandwidth that is available at the sense-amplifiers of a memory [7]. C•RAM is implemented by matching a processing element (PE) to the pitch of one or more columns of memory.

The typical C•RAM PE is bit-serial and is constructed from a set of three registers and a 256-function ALU. The ALU uses an 8-bit opcode (instruction) generated off-chip to perform the 256 possible ALU functions. All PEs receive the same opcode (Single Instruction stream) as the input to their ALU and the contents of the registers selects the appropriate result. C•RAM attempts to maintain the conventional memory interfaces for SRAM or DRAM memory, except for the allowance of new pins to provide the opcodes, additional control signals and clock for the PE.

C•RAM has previously been implemented in a DRAM process, a logic process using SRAM for the memory core [8], and in a silicon-on-insulator (SOI) process using SRAM for the memory core [16].

2.1.2 Generic C•RAM Architecture

Computational RAM (C•RAM) is a SIMD (Single-Instruction stream Multiple-Data stream) architecture that was developed at the University of Toronto [8]. The architecture builds on the structure of DRAM and SRAM memory cores in order to minimize the architectural changes that would be required to implement C•RAM into a memory core. C•RAM has been implemented in a DRAM IC process, and logic IC processes.

The architecture consists of 1-bit processing elements (PEs) constructed as a linear array. The 1-bit PE contains a 256-function ALU and 3 registers which act as operands to the ALU. C•RAM has been constructed using DRAM and SRAM as the main memory of the PEs. Each PE has access to its own local memory through the means of a global address (uniform addressing). C•RAM is a SIMD architecture because all the PEs are supplied with the same instruction (Single-Instruction stream), although each PE operates on its own data (Multiple-Data stream). C•RAM can perform point-to-point communication where there is one sender and one receiver. For C•RAM the point-to-point communication is patterned so that all PEs shift left or right to their nearest neighbours, and when distances greater than one are required, the shift operation is performed multiple times. C•RAM also has the capability to broadcast data and combine data through a global bus (broadcast bus), which has been implemented as wired-OR or wired-AND. To improve scalability, C•RAM was designed with a scalable left-right interconnect and broadcast bus to allow for scaling of the processor array by connecting multiple chips together.

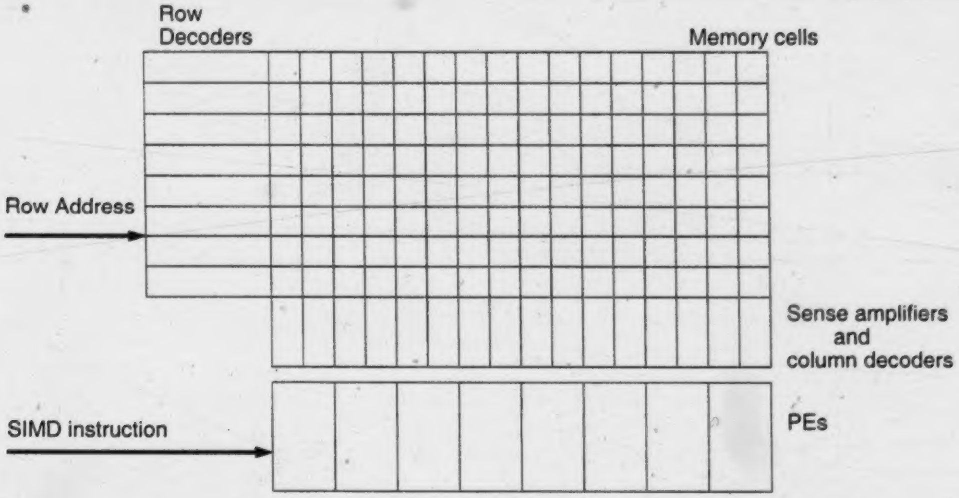


Figure 2.1: C•RAM Array - Memory cell array incorporating PEs [8]

The idea behind C•RAM was to take advantage of the bandwidth that is available from the sense amplifiers of a memory. By increasing the size of memory and allowing each column or several columns of memory to be tied to a PE, the processing power increases with the amount of columns of memory or banks of memory that are introduced in each generation. C•RAM was also designed to replace standard memory, so that the user only has to buy one type of memory. The advantages of C•RAM can best be seen in areas of computing where there is a high level of parallelism. The PEs are directly integrated into the memory and data is stored in each column of memory. For C•RAM to be useful, there needs to be enough computing that can be parallelized to take full advantage of the C•RAM architecture. When the computing parallelism is low, C•RAM is likely to be slower than a uniprocessor system, since C•RAM is only a 1-bit serial processor and uniprocessors are typically 32-bit or 64-bit processors. Another advantage of C•RAM is that it is a processor-in-memory (PIM), so it is a system within itself and is able to operate at lower power than a complete uniprocessor system. In addition, the C•RAM chip is small and therefore can be implemented in a hand-held system and still maintain the advantages of high-speed computing and low power usage.

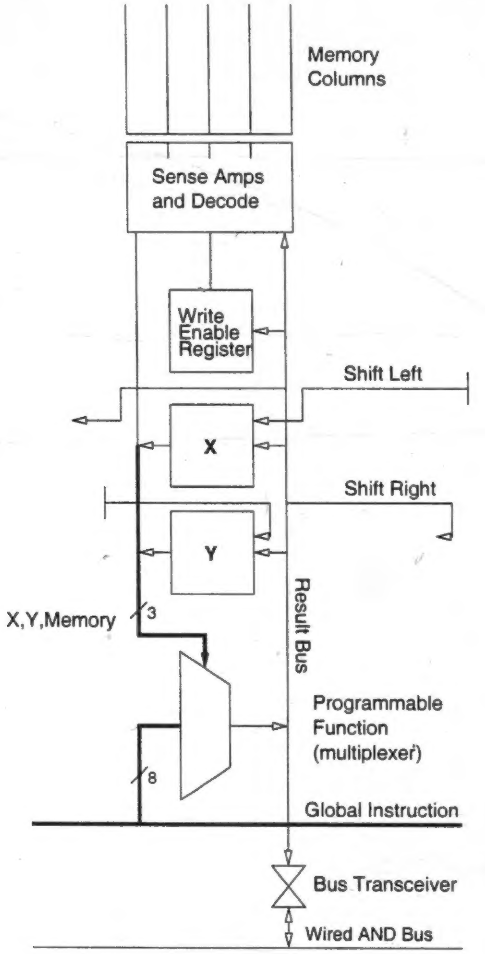

Memory Columns
Sense Amps and Decode
Write Enable Register
Shift Left
X
Shift Right
Y
X,Y,Memory
3
Result Bus
Programmable Function (multiplexer)
8
Global Instruction
Bus Transceiver
Wired AND Bus

Figure 2.2: C•RAM PE - 256-Function PE including interconnection [8]

Despite this advantage, there is the disadvantage, that the PE does not fit into the pitch of two bit-lines in a DRAM process. Additionally, the logic (PE) does not scale the same way in a DRAM process; therefore, with each new generation of processing technology, it may be necessary to re-design the PE. In a DRAM process the PE is usually built to fit the pitch of eight bit-lines in the DRAM memory. This adds additional costs to development of a new C•RAM with each generation.

2.1.3 3DSOI C•RAM

At the University of Alberta, an implementation of C•RAM was built using a silicon-on-insulator (SOI) process supplied by MIT Lincoln Labs [16]. The process from MIT allows for the stacking of multiple dies (wafers), since one of properties is that the substrate is actually an insulator rather than a conductor. This allows for the etching of holes through the die and filling the holes with a conductive layer that can tie signals from one die to a signal in another die. The implementation of C•RAM in SOI was done with the concept that identical dies can be stacked, and that the processor array would scale linearly by the number of processing elements on a die.

2.1.3.1 Memory Organization

The memory organization of 3DSOI C•RAM is similar to the generic C•RAM architecture. The memory core of the 3DSOI C•RAM is SRAM and is placed in a grid pattern. Since the memory core is SRAM, the interface to the memory is similar to a SRAM interface with row and column address being supplied together. For C•RAM, the ability to turn on only the row for operations is included so there is an enable signal for the row and column address. This allows the user to read and write data to the row and column of memory by asserting both enable signals.

The memory consists of one bank of memory with 256 PEs for the bank. The bank consists of 256 rows of memory cells for each PE, and a single column of memory cells for each PE. This gives a result of 64Kb per C•RAM die, but with the MIT process two dies can be stacked resulting in a 128Kb SRAM with 512 PEs per chip. To relax the pitch that the PE needed to be, the PEs were distributed above and below the memory array. This corresponds to each PE pitch being matched to two columns of memory; however, there is a PE above and below the memory array, resulting in two PEs being constructed in the pitch of two columns of memory.

The memory organization for the 3DSOI C•RAM is shown in Figure 2.3.

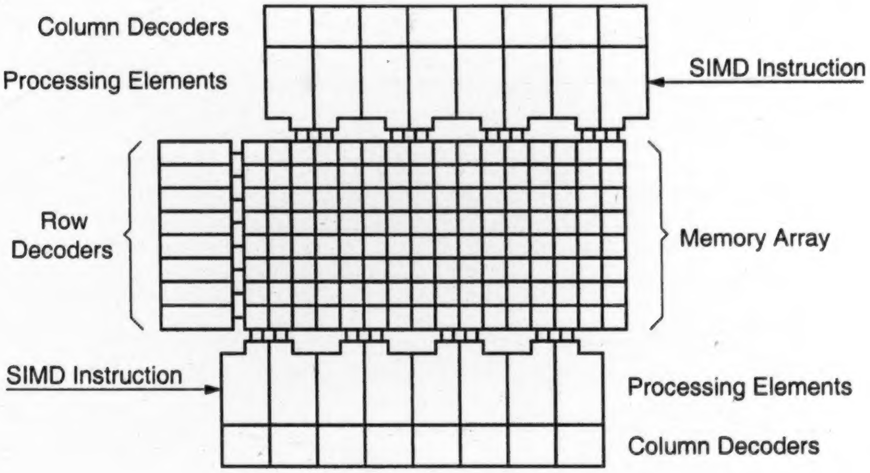

Column Decoders
Processing Elements
SIMD Instruction
Row Decoders
Memory Array
SIMD Instruction
Processing Elements
Column Decoders

Figure 2.3: 3DSOI C•RAM Organization [16]

2.1.3.2 Processing Element

The processing element (PE) for 3DSOI C•RAM is similar to the PE of the generic C•RAM architecture. The PE is a bit-serial processor with three registers and a 256-function ALU. The ALU remains as a 8-to-1 MUX which can perform 256 functions, but the MUX was built using pass-transistor logic. The PE has two additional registers: one is used when performing conditional operations, and the other is used for maintaining communication in the event of a failure in the PE or in the column of memory associated with the PE.

2.1.3.3 Communication

The communication scheme used in 3DSOI C•RAM is a left-right 1-bit shift network, where each PE shifts the ALU result to its left or right neighbour, depending on the control signals. This interprocessor communication allows for all PEs to send data to other PEs one or more PEs away, and to retrieve data from other PEs that are sending their data at the same time. This can be thought of as patterned communication, since all PEs send data the same distance simultaneously, as seen in Figure 2.4.

There is also a broadcast circuit that allows a single PE to broadcast a result

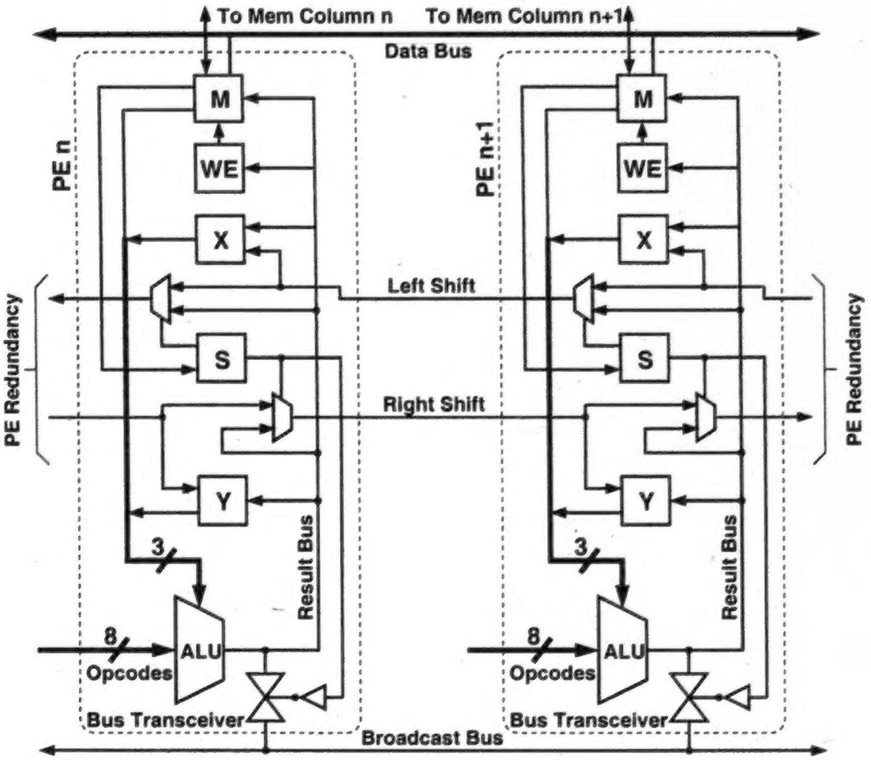

To Mem Column n
To Mem Column n+1
Data Bus
PE n
PE n+1
M
WE
X
Left Shift
S
Right Shift
Y
PE Redundancy
PE Redundancy
Result Bus
Result Bus
3
3
8
8
ALU
ALU
Opcodes
Opcodes
Bus Transceiver
Bus Transceiver
Broadcast Bus

Figure 2.4: 3DSOI C•RAM Processing Element [16]

to one or more PEs. This broadcast circuit also has the ability to combine the results of multiple PEs (global wire-OR or wire-AND), so that all PEs communicate simultaneously with each other to determine whether or not one or more PEs are looking at the same data. For example, this broadcast network can be used to find a minimum or maximum data value among the PEs.

2.1.3.4 Redundancy

In 3DSOI C•RAM, the redundancy technique was used to shift the problem down the array. If one PE or column of memory failed, then it would set its skip register to '1', indicating that it should not participate in the shifting of data due to a fault associated with the PE. This way the chip can still be usable, even if at the end there are only 128 PEs out of 256 PEs that are usable. In Figure 2.4, the redundancy circuit is shown for the PE. This redundancy circuit allows the bypassing of one or more PEs. More information about C•RAM redundancy can be found in Section 2.4.3.

2.2 Silicon-on-Insulator

2.2.1 Overview

The Silicon-on-Insulator (SOI) CMOS technology process has emerged as a potential alternative to the conventional bulk CMOS technologies that are currently in use. The advantages of SOI are related to the feature of device islands (transistors) being dielectrically isolated from each other and from the underlying substrate. Due to this feature, there is a reduction in the junction capacitance of transistors, CMOS latch-up is eliminated, short-channel effects are enhanced, and the immunity to radiation induced soft errors is improved [5]. The lateral isolation of device islands allows for more compact designs, since there is no need for wells or inter-device trenches [6].

Since SOI is an emerging technology, difficulties in manufacturing and design at the device and circuit level are common. Despite these difficulties, an SOI design can - and most often will - exhibit reductions in power consumption when operating at the same frequency as an equivalent conventional bulk CMOS design. Alternatively, an SOI design can operate at higher frequencies than the CMOS design for the same power consumption [5].

SOI processes can be classified by active area depth into fully-depleted (FD) and partially-depleted (PD). There are advantages and disadvantages to both device technologies. In Table 2.1, the characteristics of FDSOI and PDSOI that were considered during the design of the Alpha microprocessor are shown [11]. The results show six reasons to choose the FDSOI, and six reasons to choose PDSOI.

In the table poor indicates that the process gives poor results when the condition is examined. For example, the leakage current for FDSOI is poor this means that FDSOI has higher leakage currents than the PDSOI process. Likewise, for FDSOI, the stability of a transistor without a body contact is good indicating that FDSOI

Table 2.1: PDSOI versus FDSOI [11]

	PDSOI	FDSOI		PDSOI	FDSOI
Performance	Good	Good	Ease of Manufacture	Good	Poor
Design Compatibility	Poor	Good	Breakdown Voltage	Good	Poor
CAD Environments	Poor	Poor	Leakage Current	Good	Poor
Stability w/ contact	Good	Fair	Transconductance	Good	Poor
Stability w/o contact	Poor	Good	Short Channel Effect	Poor	Good
Operation Voltage	High	Low	Body Contact	Good	Poor
History Dependence	Poor	Good	Body Effect	Good	Good
Parasitic Bipolar Effect	Poor	Good	Self-Heating	Poor	Poor

still works well when transistors have a floating body. Floating body refers to case where the device well of the transistor is not connected to a voltage source, but rather floats as the transistor switches. Good is an indicator of a positive reason to choose the process, while poor or fair are indicators not to choose the process.

Choosing the SOI device type to use will depend on which reasons are most important for a specific design. While working with FDSOI may allow lower operating voltages, manufacturing FDSOI is more difficult since the silicon layer is thinner than that for PDSOI. The self-heating of PDSOI and FDSOI is a result of the buried oxide thermally insulating the transistors in the design, causing heat to build up.

2.2.2 MIT Lincoln Labs - 3D SOI

MIT Lincoln Labs has developed a Fully-Depleted (FD) Silicon-on-Insulator (SOI) technology [3, 4], where multiple wafers can be stacked to form a three-dimensional chip. This development will allow designers to look to another dimension when building chips, so that the planar area of a chip can now be used more efficiently. The stacking ability is an advantage in that separate circuits can be built on separate wafers. Additionally, the interconnect wires will not have to traverse across the chip, but can now run through the die. Previously, sending a signal across the chip would require a wire length greater than $1000\mu m$, resulting in long signal delays, but now the same signal can be connect through the chip in less than $10\mu m$.

The MIT process involves three wafers. The first wafer is a photo-diode layer, and the second and third wafer are circuit wafers. The current process is limited to two circuit wafers and three metal layers per circuit wafer. The wafer stack is created by placing the second wafer upside down on the photo-diode layer, where the photo-diode layer is first covered with a wafer bonding agent. Then the bottom of the second wafer (handle) is cut away to minimize the wafer height and the length of the vertical conductor. The wire through the die is created by etching a hole through the die, and filling the hole with metal. The holes are etched and filled forming the vertical conductors, also called 3D vias. For MIT process, the 3D via connects metal between dies, while a normal via connects metals within the dies. The vertical conductor is formed when a hole is etched through the wafer, where the third metal has a donut hole for the etching mechanism to cut through until the first metal of the wafer below is contacted, stopping the etch. After the etching is

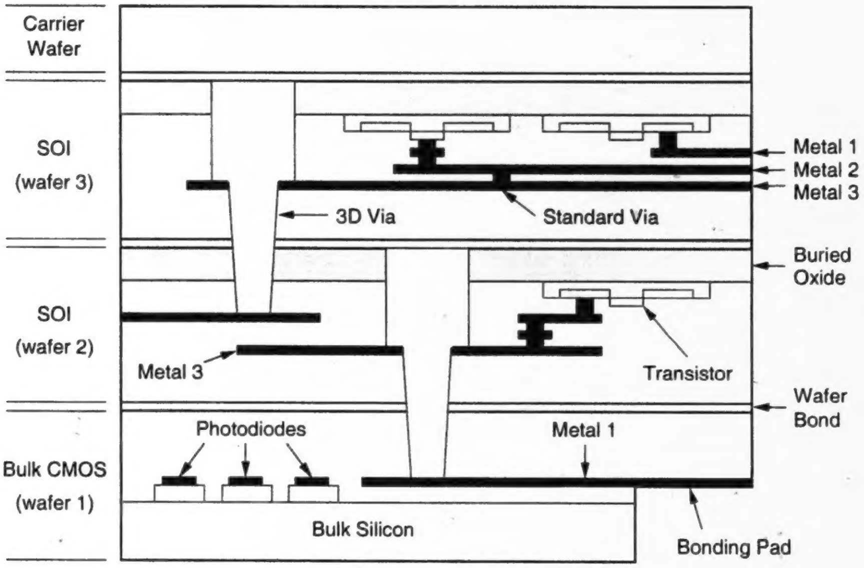

Carrier Wafer
SOI (wafer 3)
SOI (wafer 2)
Bulk CMOS (wafer 1)
Metal 1
Metal 2
Metal 3
3D Via
Standard Via
Buried Oxide
Metal 3
Transistor
Wafer Bond
Photodiodes
Metal 1
Bulk Silicon
Bonding Pad

Figure 2.5: MIT Lincoln Labs 3D SOI Stack

done, the holes are filled with tungsten, using chemical vapour deposition (CVD). The wafer is then polished to provide a smooth surface for the bonding agent to be applied. The same process is used for the third wafer. Once the third wafer is bonded and etched, a handle is applied to the bottom of the third wafer to add stability. Then the stack is flipped over and the bond pads are etched.

MIT currently has a $0.18\mu m$ Fully-Depleted Silicon-on-Insulator process that can be used to create a three-dimensional stack. When creating a three-dimensional stack, the process is limited to three metals, instead of the five metals that are available for their standard $0.18\mu m$ FDSOI process.

2.3 Grid Based Processors

In this section, some of the grid-based communication SIMD multiprocessor architectures are described. The descriptions will include the number of processing elements (PE) in the architecture, the features of the processing elements (PE), how the architecture was constructed, the inter-processor communication network, and the type of redundancy used.

2.3.1 Network Topologies

When discussing the grid-based processors it would be worthwhile to look at some of the different network topologies that can be used for grid-based networking. In Figure 2.6, some of the network topologies are shown that can be used for processors organized in a two-dimensional grid. For a two-dimensional grid, each processor is typically able to communicate with its four nearest neighbours. This network topology is typically called a NEWS (North-East-West-South) network, where each processor communicates in one of four directions. The NEWS network is the basic network topology for processor communication between processors organized in a grid. In general, EW is typically referred to as left-right communication between neighbours. The toroid interconnection network is an expansion on the NEWS network, where all the boundaries of the two-dimensional grid are

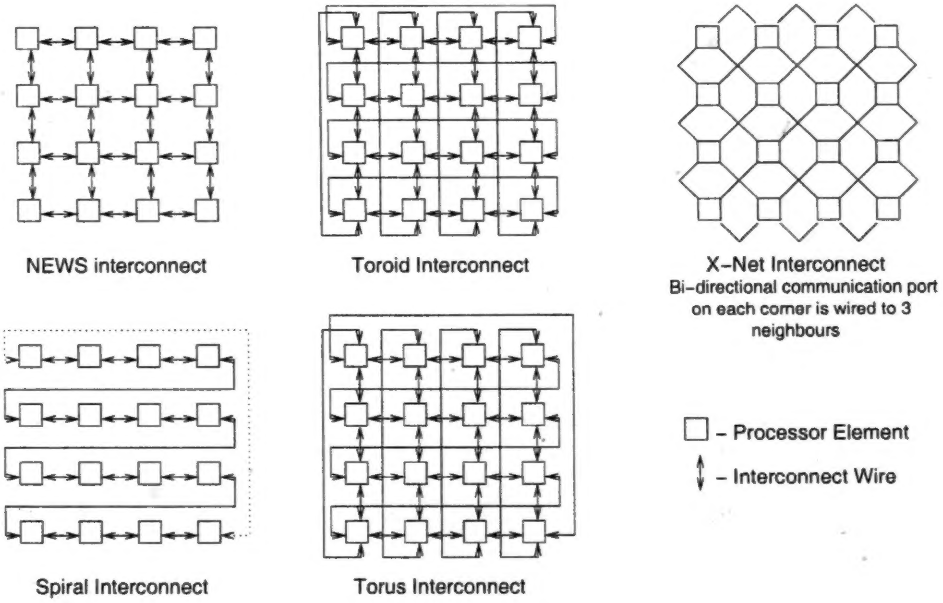

NEWS interconnect
Toroid Interconnect
X-Net Interconnect
Bi-directional communication port on each corner is wired to 3 neighbours
Spiral Interconnect
Torus Interconnect
- Processor Element
- Interconnect Wire

Figure 2.6: Network Topologies [8, 2]

connected using torus interconnects. The toroid designation comes with the understanding that all dimensions are connected using a torus interconnect. The spiral interconnect is the easiest way of creating a one-dimensional string of processors out of a two-dimensional grid of processors. This spiral interconnect allows left-right interconnections between all processors creating a complete shift chain for left-right communication. The torus interconnection network can be composed of only a torus connection for one dimension, and may include a spiral interconnect for the other dimension. The torus designation is typically used when not all of the dimensions use a torus to connect the boundaries of the grid. The X-NET interconnect is a different type of interconnect, where each processor is tied to more than one processor at a time. All processors broadcast their data out over one of the four wires that emanate from the processor, and each processor listens to one of the three remaining wires. In this way the interconnection wires are shared and allow for more flexibility in the way that communication is performed. In the following section, one or more of these topologies will be used for each of the grid-based processors. It can also be noted that many of these network topologies can be extended to three-dimensional meshes to produce similar patterns of communication.

2.3.2 Implemented Grid Based Processors

ILLIAC IV

The ILLIAC IV was the first operational processor array, and was built in 1972 [2]. When it was built, it was designed to have four 64 PE quadrants, but the first system only contained one quadrant. The ILLIAC was designed with each PE having access to a 2048 64-bit word memory. The memory was fully accessible by the control unit, but each PE could only access its own memory using uniform addressing. Each of the PEs was designed with its own arithmetic circuitry and 4 64-bit registers, which were used for holding shifted data, accumulator data, operand data and temporary data.

The 64 PEs were organized into an 8x8 processor grid (two-dimensional) with each processor being able to communicate to its four nearest neighbours using the NEWS network topology network, where each processor communicates in one of four directions. The inter-processor communication (shift) network topology for the ILLIAC is described as a torus, where the top and bottom processor of each column in a two-dimensional grid are connected together. The left processor is connected to the right processor of the previous row in a two-dimensional grid. This type of connection is typically called a spiral network topology, since the data would be shifted in a spiral nature around the processor array. Since the bottom-right processor is connected to the top-left processor, the East-West network topology is also a called a torus. No mention was made to the type of redundancy, leading to the belief that no redundancy of either PEs or memory was done.

DAP

The ICL DAP (Distributed Array Processor) was first implemented in 1976, with 1024 PEs having 4Kb of memory per PE [8, 12]. The PE was a 1-bit processor containing a full-adder and 3 registers. The system was constructed using medium scale integration TTL components and standard memory chips. The inter-processor communication network topology for the ICL DAP was the NEWS network, where each processor communicates with its 4 neighbours on a two-dimensional grid. This communication was done using a 4-to-1 multiplexer in each PE to select the network output of one of the four adjacent PEs as the PE's input.

The DAP architecture was further developed by Active Memory Technologies Ltd. (AMT) into the DAP 500, 510C and 610C. These were developed as a SIMD "back-end processor" to work with a VAX minicomputer or SUN workstation host [8]. The DAP 500 was AMT's version of the ICL DAP; it contained a 32x32 PE two-dimensional grid with 64 PEs per integrated chip. For the DAP 510C and DAP 610C, AMT added 8-bit math co-processors for each PE to improve arithmetic performance; however, these were located on separate chips. The DAP 510C was an array of 32x32 PEs, and the DAP 610C was scaled to 64x64 PEs. The DAP 500 used parity to increase reliability, duplication of the entire operation, and master/slave comparison as means of redundancy [20].

MPP

The Massively Parallel Processor (MPP) developed by Goodyear Aerospace for NASA Goddard Space Flight Center in 1981 contains 16384 PEs organized in a 128x128 processor plane capable of two-dimensional communication between processors [2, 8]. Each PE in the MPP had access to 1K bits of memory, and the local memory is accessed through uniform addressing. The MPP was equipped with a 16-MB staging memory which was used to reformat and buffer the data for the processor plane. Each PE is a bit-serial processor containing a full adder, 6 registers, a programmable-length shift register, a boolean and routing-logic unit, comparison logic, and a local data bus. The system was built with 8 processors per chip and with off-chip memory for each PE. The memory for the PEs was organized in 128x128 1-bit per PE planes. The 16384 PEs were organized into a 128x128 processor grid (two-dimensional) where each processor could communicate between its four nearest neighbours. The interconnection topology is similar to the ILLIAC IV in that it uses a NEWS interconnection topology. MPP added the ability to transform the interconnection into a toroid, where the edge processors shift to the processors on the same row or column at the other edge of the grid, forming horizontal and vertical cylinders. It also has the ability to form a spiral, closed spiral and a spiral toroid. No redundancy was found for the MPP architecture.

Connection Machine

The Connection Machine (CM) is an architecture that was developed by Thinking Machines Corporation in 1985 and, like the architectures above, is a processor array [2, 8]. Of all the Connection Machines implementations, the CM-1, CM-2 and CM-5 architectures are discussed most often. The CM-1 was implemented with 64K bit-serial PEs, with each PE having 4K bits of memory. The PE of the CM-1 consisted of dual Boolean units (256 functions), 8 general-purpose registers, and a selector for selecting the source operand for the Boolean unit. The CM-1 was constructed using commodity DRAM for main memory, each processor chip containing 16 PEs with an autonomous hypercube router and a PE control unit. The CM-1 contained two interconnection topologies, where the first was the standard NEWS network where each PE could communicate with its nearest neighbour. The second topology was an autonomous hypercube network, where two PEs were separated by no more than 12 hypercube wires. No redundancy was found for this implementation.

The CM-2 is similar to the CM-1, except that the NEWS network was dropped from the interconnection topology. The system was changed to have a 64-bit floating point processor for every 16 PEs. Within a group of 32 PEs, each PE can access another PE's local memory. The memory for the CM-2 was expanded to allow 1Mb per PE; this was still done with commodity DRAM. The interconnect was split into two levels, where in the first level the PEs, in a group of 16, communicate by writing directly into each other's memory by means of a flipper network (butterfly network). The next level involves the hypercube of dimension 12 (a 12-cube), where each PE group occupies one vertex of the 12-cube. No redundancy was found for this machine as well.

The CM-5 was built as MIMD machine using commercially available microprocessors (SPARC), instead of custom chips [18]. For the CM-5, the network and data interface are decoupled from the microprocessors so that the networks are not relying on the microprocessors in order to perform network functions. The CM-5 was designed to be resilient in the presence of faults, despite the specific network topology.

EXECUBE

The EXECUBE architecture built in 1994 consists of 8 16-bit PEs with 4.5Mb of DRAM memory. The PEs can operate in SIMD mode, with each PE receiving a broadcast instruction, or in MIMD mode, with each PE fetching their own instruction [8]. Since the processor array can operate in MIMD mode, it was required that each PE has its own address by which to access its memory, which is also called autonomous memory addressing. Each PE has access to two 32K 9-bit word DRAM arrays, where the 9th bit is parity. The processor array is built in a 4Mb DRAM process with 8 processors implemented using ”sea of gates” semi-custom logic.

The PEs are connected in a cube formation with each PE having four point-to-point links for interprocessor communication. This is equivalent to one PE at each point of the cube. For off-chip communication, 8 links are made available instead of the required 24 for a full three-dimensional mesh. No processor redundancy was found for this architecture. Since the DRAM was built using a commodity DRAM macro, memory redundancy techniques are more than likely incorporated into the DRAM.

2.4 Redundancy

Redundancy is a method where the reliability and yield of a design can be improved to the point of reducing the number of manufactured chips that are rejected. Redundancy is used to eliminate or limit the impact that defects in the die may have on the bonded chip. In this section the types of redundancies that have been used previously will be discussed briefly.

2.4.1 Logic Redundancy

When implementing redundancy for the means of increasing yield, maintaining or preserving functionality of a logic-based chip, it may be necessary to create a replacement block or element. Adding redundant logic will increase the area of the block. For C•RAM, if the redundant logic was added to each PE, the area of the entire PE would increase. The selection of which piece of the PE will be replicated to add redundancy must then be made to ensure reliability.

Another method of redundancy would be to create a redundant PE instead of adding a redundant element to the PE. This will cut down on the area that is used for redundancy, but the number of redundant PEs to use becomes the factor in improving the yield.

2.4.2 Memory Redundancy

The use of redundancy in designing is an effective method of improving the reliability, increasing the production yield, and reducing the cost per bit for memory chips [13]. When working with memories, the use of redundancy is needed because the density of the design is high and the probability of a defect or failure in the memory array is also high.

2.4.2.1 Conventional Techniques

In memories, the redundancy techniques replace defective memory elements using on-chip spare elements. The defective elements are usually word lines or data lines. For word-lines, there are N spare word-lines with N address comparators. The address of the defective word-lines are programmed into the address comparator during wafer testing, and one of the spare word lines is activated anytime the input address matches one of the defective addresses. The programming elements are usually poly-silicon fuses, which are blown by means of a laser beam or a pulsed current.

For word-line (row) redundancy, there are a number of ways to replace the defective word-lines. When using sub-arrays, two sub-arrays share a common sense amplifier so only one of the word-lines will be activated. If there are sub-arrays in the chip, which is common, then the defective word-line address can be applied globally to all the sub-arrays simultaneously. In this way all the sub-arrays replace the word-lines with the same spare word-lines, and this reduces the number of comparators. The word-lines can also be replaced individually, which results in the needed for less spare word-lines per sub-array [13].

Another technique involves activating the spare word-line and the defective word-line simultaneously. This involves splitting the sense amplifier between two sub-arrays, and programming the spare word-line onto the sub-array that does not have the defective word-line. This results in no access penalty, since the defective and the spare word-line results are independent. The data is then selected from the sense amplifier containing the spare word-lines data.

In Figure 2.7, the conventional technique used for data line (column) redundancy is shown. Spare columns of memory can be used to replace a defective column of memory in a similar way to how a word-line (row) is replaced. For column redundancy, columns can be selected using a multiplexer to determine which column will be used. The multiplexer selects whether the defective column or the spare column is to be used [9].

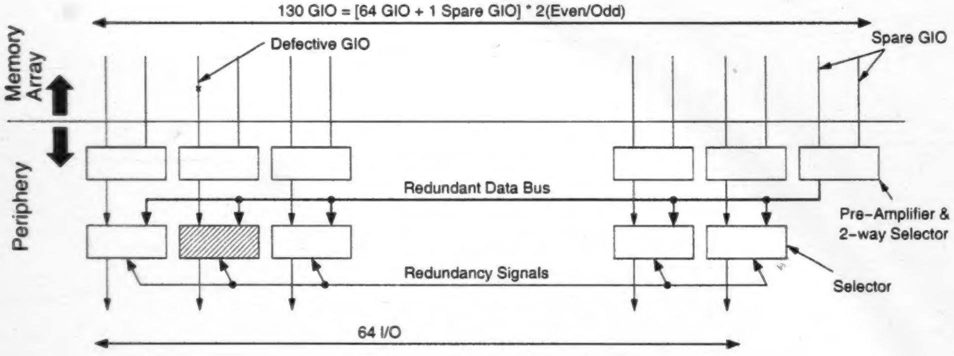

130 GIO = [64 GIO + 1 Spare GIO] * 2(Even/Odd)
Defective GIO
Spare GIO
Memory Array
Periphery
Redundant Data Bus
Pre-Amplifier & 2-way Selector
Redundancy Signals
Selector
64 I/O

Figure 2.7: Conventional Column Redundancy Architecture [9]

2.4.2.2 Embedded Memory Technique

In [9], a column redundancy technique for an embedded DRAM core is presented. The method of column redundancy is used to shift the column addressing over by one column, as shown in Figure 2.8. To accomplish this the data lines (bitlines) are connected to pre-amplifiers, then the output of the pre-amplifiers is selected. This keeps the column addressing uniform, and results in no access penalty when compared to the conventional column redundancy scheme.

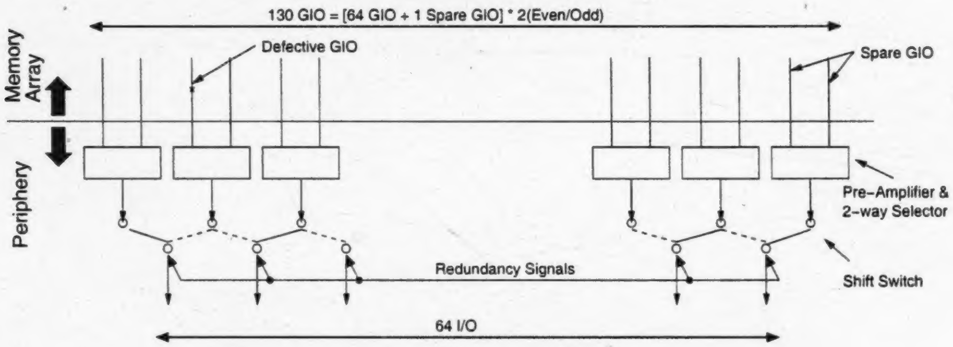

130 GIO = [64 GIO + 1 Spare GIO] * 2(Even/Odd)
Defective GIO
Spare GIO
Memory Array
Periphery
Pre-Amplifier & 2-way Selector
Shift Switch
Redundancy Signals
64 I/O

Figure 2.8: Shift Column Redundancy Architecture [9]

2.4.3 C•RAM Redundancy

As C•RAM is a memory variant and memory redundancy typically adds rows and/or columns of memory, it makes sense to add redundant PEs instead of adding redundancy to the PE. When doing this, however, the redundant PEs and columns of memory will have to accommodate failures in both the memory array and the PEs. Since C•RAM has a shift network where the PEs are connected in order, the redundancy mechanism must preserve the functionality of the shift network [8, 16].

2.4.4 Grid Based Redundancy

In [14], a dynamically reconfigurable interconnect for array processors is presented. The reconfigurability of processor arrays is important because it allows for efficient execution of different algorithms and for the isolation of faulty processors. The interconnection network uses SRAM cells to configure the interconnect switches to bypass faulty processors or to form different interprocessor network topologies. The use of SRAM allows for the dynamic reconfiguration of the processor array in the field and different algorithms that require other network topologies. The reconfiguration algorithm allows for elimination of the minimum number of rows/columns that cover all of the faulty cells, while maintaining the desired network topology. Each interconnection switch is configured by a SRAM storage cell, requiring many cells to be used to create the interconnection networks.

Redundancy for grid-based logic ICs is more logical considering its regular structure. In [17], the author explains how it is more suitable to incorporate redundancy in a logic circuit with a regular structure, as opposed to a microprocessor (random IC) which is more complex than a memory IC. The author also illustrates three reconfiguration approaches for grid-based logic. In one approach a failure in a row or column results in the complete removal of the row and column containing the fault; see Figure 2.9(a). The second approach, Figure 2.9(b), steps over faulty logic blocks in the array. The third approach, not shown, is to perform interstitial redundancy where a number of PEs are attached to a redundant element. In the author's example each logic element is connected to two redundant elements, so that

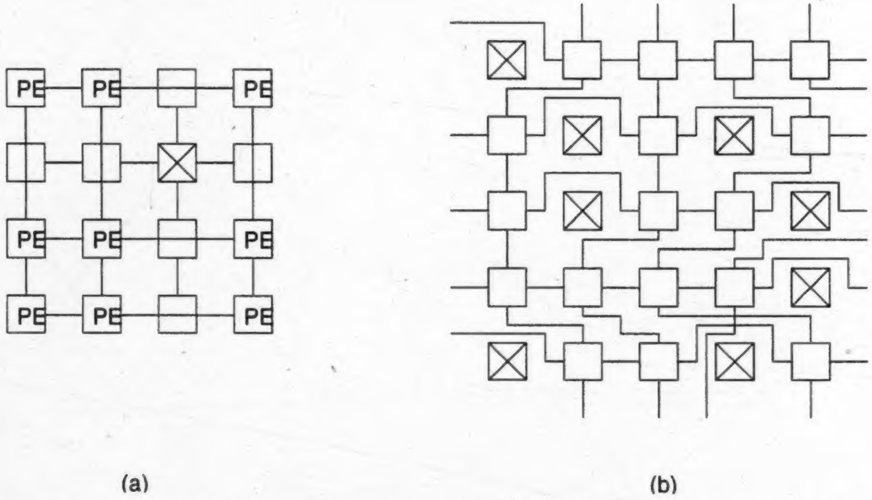


Figure 2.9: Grid Based Redundancy Approaches [17]

the faulty logic blocks can be more efficiently and effectively replaced to maintain functionality. In Figure 2.9(b), the author was showing a simple reconfiguration scheme that adds spare columns to the array. The processors were re-indexed in the rows to skip faulty processors. Then the vertical connections were made between the rows. This scheme requires a complex switch and interconnection structure to support reconfiguration.

2.4.5 Altera's 2D Programmable Redundancy

For Altera programmable logic devices, redundancy is provided by the addition of spare columns and rows of logic blocks and switch boxes. The device is able to bypass a column or row containing one or more defective logic blocks, and switch in a spare row or column of defect-free logic blocks [10]. This redundancy implementation bypasses an entire row or column of logic blocks, and switches to the defect-free redundant row or column. The use of multiplexers at the inputs and outputs of the logic blocks allows the bypassing of the defect logic blocks. The method of redundancy shifts the configuration data over to an adjacent column of defect-free logic blocks by means of a datapath. This can be extended to defective switch boxes, allowing for the replacement of a defective row or column of switch boxes.

2.5 Summary

In this chapter, a few of the available network topologies for two-dimensional PE grids were defined. These network topologies need not - and should not - be confined to only two-dimensional PE grids. In the next chapter, a dynamically reconfigurable three-dimensional PE grid is designed that makes use of some of the two-dimensional network topologies that were examined. The reasons for redundancy in C•RAM were discussed, and in the next chapter the redundancy is extended to include another dimension of PEs. This design focuses on the creation of a three-dimensional mesh of PEs, and the addition of redundancy along the z-axis of the cube is proposed. The following chapter details the architectural changes required to design a fault-tolerant, dynamically reconfigurable 1D, 2D and 3D Communication Network for C•RAM.

Chapter 3

Architectural Concepts

The development of the generic C•RAM architecture has led to a highly parallel and compact SIMD machine capable of high speed computation on highly parallelizable code. Previous generation C•RAM architectures have had a linear communication network, and programming algorithms that involved multiple dimensions of shifting required complex programming to perform the shifts. If the number of shifts in a parallel program are high due to the large diameter of the network, then the majority of the processing time can be taken up by shifting. To speed up multiple dimensions of shifting, an interconnection network that selectively supports one-dimensional, two-dimensional, and three-dimensional shifting is proposed and explained in this chapter; design specifics are in the following chapter. In addition, due to the high overhead of I/O when performing real-time parallel computing, a bank-style memory architecture for simultaneous I/O and computing is also proposed. Since these are new aspects of the C•RAM architecture, the PE needs minor modifications to accommodate the changes to the architecture. A novel processor redundancy architecture, which maintains functionality of the proposed communication networks, is also developed.

3.1 Memory Array

For the C•RAM architectures of the past, the external datapath has been small due to the fact that data is re-organized in the memory to accommodate a large number of bit-serial processing elements (PE). In C•RAM, each data element (i.e. integer) is required to be stored per PE (column of memory), where as in standard DRAM, the data elements are stored in rows across multiple columns of memory. For this reason the external datapath of C•RAM has usually been kept small to reduce the amount of processing that needs to be performed on the data before supplying it to the C•RAM chip. The restriction on the external datapath means that time to load or store (save) the data of the program can be quite extensive, and a large amount of computing time is taken up by data transfer. One way to improve the speed of data loading is to implement a 'corner-turning' cache, where data is loaded into the cache and then transposed to allow multiple columns to be written simultaneously, allowing for a wider datapath into the memory. This would reduce the amount of time used for data transfer.

The above method does not reduce the effects of data transfer enough to make C•RAM viable enough as a parallel processing system. Typically, computation cannot occur until the entire array of data has been loaded or stored. By performing computations and external memory accesses individually, the amount of computing that can be performed per second is reduced. For example, trying to MPEG encode a TV signal or digital camcorder signal in real-time requires that the number of frames per second be a certain value; about 30 frames per second for TV, and 24 to 30 frames per second for the digital camcorder. If, however, the amount of time taken to load the memory array and store the encoded data to disk is 33.3ms, then the amount of time remaining to perform computations during one second is zero. This would actually result in a decrease in the number of frames per second being encoded; therefore, frames would be lost. If the processing elements, however, could be performing calculations on the previously loaded data while data is being transfered for the next set of calculations, then the problem of having zero time available for computations is removed, since computations are performed during data transferring.

Therefore, the method of C•RAM I/O improvement being described in this chapter reduces the amount of computing time being wasted by data transfer by allowing the data to be loaded or stored while the processing elements are performing computation: allowing data transfer and computation to occur in parallel. The architectural enhancement to the memory is the implementation of loading or storing external data during computation. This architectural enhancement can be combined with the 'corner-turning' cache to improve the overall performance of the C•RAM I/O.

3.1.1 Multi-bank Design

To improve the I/O performance of C•RAM, a multi-bank memory design is proposed that allows computation and external memory access to occur simultaneously, in parallel. This reduces the amount of time that is stolen from computation when external load or store operations need to be performed. It can be shown how the computations and memory access for a C•RAM chip containing 512 PEs with 1024 bits of memory per PE are related. Assuming that the external access cycle is 8ns, the internal access cycle is 4ns and the ALU operates at 3ns. It will be assumed that one in three ALU operations requires two internal accesses to write and read the M register value. From these assumptions the external reading and writing of the memory array will require:

$$t_{external} = 2cycles \times 512PEs \times 1024bits/PE \times 8ns/bit = 8.389ms$$

Using the time required for external reading and writing of the array, the number of PE operations that can be performed in that amount of time can be extracted. Since it was said that one in three ALU operations requires two internal accesses, the time for the accesses will be distributed over the three ALU operations. From the assumptions, the number of ALU operations that can be performed in the same time is:

$$num_{ALUOps} = 8.389ms/(3ns + (8ns/3)) = 1480342$$

Therefore, if the external accesses can be performed during the computation cycle, then it is possible to perform 1480342 ALU operations in place of the external accesses. By hiding the external accesses in the computation cycles, it is possible to read and write the memory externally a maximum of 119 times per second; thus, twice as many calculations can be performed per second. If the external accesses were to be performed after computation and the computations take the same amount of time as external accesses, then the complete cycle (read, write and compute) can only be performed 59 times per second.

This multi-bank design would be ideal when the C•RAM is used as an MPEG encoder. For MPEG encoding, the need for near real-time computing requires the ability to load the next frame into memory, while computation is performed on the current frames. In MPEG encoding, the encoding is dependent on the current frame and the previous video frames. In this way, the number of frames encoded per second can be maximized, since there is no need to wait for all the data to be loaded entirely before proceeding to the next computation. This is done using a four bank strategy: one bank can contain the previous frame; one bank can contain the current frame; one bank contains the result; and another bank can be loaded with the next frame in the sequence while the processing is done with the other three banks. The load or store can be partially or fully hidden in the computation, allowing for more computations, and therefore frames, to be done per second, since there is no need to wait for complete loading and storing of the data externally. In Figure 3.1, the architecture is simplified to two banks, but can be easily extended to more than two banks, with minor changes to the row decoders. There needs to be a row decoder for each bank of memory to provide full functionality and interleaving of the multi-bank architecture.

The concept used for the multi-bank design is to perform auto-addressing internally, so that the external interface always references the banks in a sequential manner. In this way, the user or programmer does not need to write program code to determine the new bank address. The memory banks for the processing element always appear as bank 0, 1, and 2, while the external datapath interface only ever sees one bank. Once the externally visible bank is finished loading and/or storing and the processing elements are ready to continue to the next phase, a signal can be applied to the chip indicating that the internal address (bank referencing) needs to

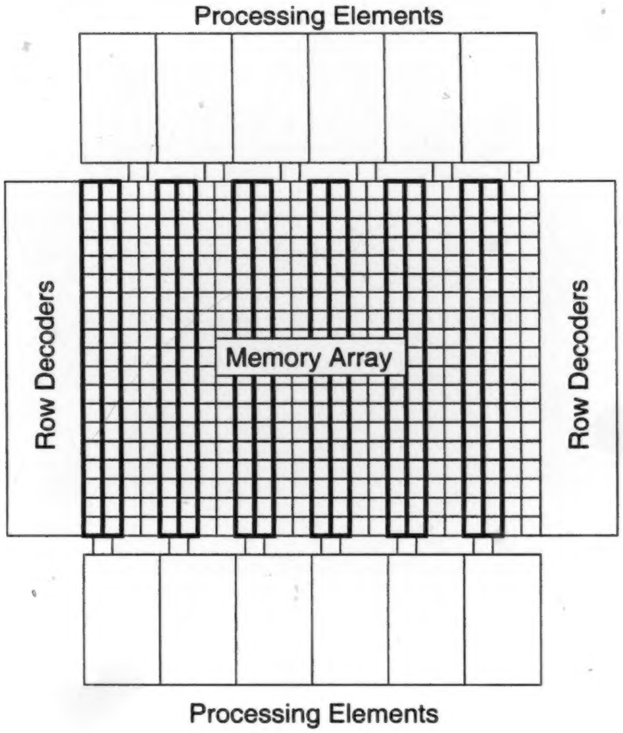


Figure 3.1: Multi-Bank C•RAM

be incremented or decremented, depending on design choice. In Figure 3.1, shows that there are two row decoders for the array. Each bank of memory should be individually addressable, and therefore, each bank requires its own row decoder. Using this architecture, the processing element and external datapath have access to both banks of memory and can address them separately. In this way the external datapath can address a different row of memory than the parallel program is accessing for the processing elements.

The multi-bank design can be built with all banks of memory being individually addressable, such that every bank of memory can access different rows simultaneously. In this way, accesses to memory can be interleaved, so that when the PE is using one bank for accessing, the other banks can be precharging in preparation for the next access. This method requires a memory controller that can properly time the control signals, unless the control signals for every bank are externally controllable. The multi-bank memory controller should still be able to perform auto-addressing such that the external interface need not keep track of the new bank addresses. Alternatively, the multi-bank design can be built with the banks being grouped together into PE accessible banks and external accessible banks. This requires that only two addresses are required for row accessing; thus, only two row decoders, and only two groupings of control signals. In this way, the PE accessible banks would all be accessed simultaneously with the same row address and control signals. This method removes the ability to interleave row accesses; however, it simplifies the row accessing for PE computation and external addressing. It also simplifies the external interface and memory controller.

By performing auto-addressing internally, the programming is always able to reference the banks as 0, 1, and 2 with no need to externally keep track of which bank of memory is actually the zero bank of memory. For MPEG encoding, the banks can always appear as result, previous, current, and next data, respectively for programming purposes because of the auto-addressing scheme. This allows the programmer to take advantage of the data dependency between loops of a program. This architecture is similar to the method of frame buffering used in video cards for graphics, where the frame buffer is always written in the same way. The data in the frame buffer is position sensitive because each data location corresponds to a position on the display.

3.2 Interconnection Network

The inter-processor connection network of previous C•RAM architectures has been a one-dimensional patterned communication network within rows of PEs, where each processing element (PE) communicates left or right to its nearest neighbour, and two-dimensional patterned communication beyond the rows. This poses a limitation on the type of algorithms that can be performed efficiently on the C•RAM architecture. Previous C•RAM generations have also had the capability of extending the communication network over many chips in order to meet the processing needs of some parallel applications. This section proposes a dynamically reconfigurable inter-processor connection network that allows multiple dimensions of inter-processor communication in an attempt to enhance the ability of C•RAM to perform more complex algorithms and to perform shifting more efficiently.

The implementation of a dynamically reconfigurable inter-processor connection network should maintain the capability of extending the communication over many chips. In this way, multiple chips can be combined to meet the processing needs of the parallel application being executed.

3.2.1 Applications of a Multi-dimensional Network

The development of a multi-dimensional network was considered to increase the commercial viability of C•RAM, since previous C•RAM designs have been somewhat limited by the interconnection network. In previous C•RAM designs, applications that extended beyond 2D required complex programming to perform shifting and storing of data. Typically, 2D applications were implemented by using the memory above the PEs, such that the PEs were the x dimension and the rows of memory became the y dimension. This organization still required some complex programming to perform shifting of data; however, not as much as for a 3D application.

The main 3D application that could benefit from a 3D mesh of processors is volumetric rendering, where each PE would represent a volume pixel (voxel). In volumetric rendering, each voxel is created by combining the density data of neighbouring pixels. For volumetric rendering, data from a MRI scan is converted to a digital image of pixels. Each pixel will represent a density based on color or shade of the coordinates. From this pixel information, a volumetric rendering can be made that will represent the images in a more readable form. To perform this, every voxel requires access to all of its nearest neighbours; however, the algorithm has been simplified so that only 6 of its nearest neighbours are used, rather than all 26 neighbours at all points of a 3x3x3 cube. Only 6 of the points are used due to the increased overhead of shifting or obtaining the data for all 26 points; nonetheless, 6 points can produce a fairly accurate result. By looking at this example, we can see that if a cube of processors can use patterned communication and can communicate in 6 dimensions, it could be possible to retrieve all 26 neighbouring points to create a very accurate volumetric rendering of the data.

The development of a multi-dimensional network that incorporates a 2D interconnection network could be used for 2D applications that have a time component to them, such that the time component would be stored in the memory rows of the PEs. In this way, the 2D shifting of data would be minimized due to the 2D interconnection network. MPEG encoding or decoding, JPEG compression, or image manipulation are some 2D applications that could possibly benefit from the implementation of a 2D interconnection network.

The benefits of a dynamically reconfigurable network can also be seen when looking at the problem of shifting for a 1D network of PEs. If we assume that 512 PEs are involved in the 1D network, and the number of shifts to the left or the right varies from 1 to 512 PEs, then a large amount of computing time will be taken up by shifting. If the PEs are not only connected as a 1D network, but also as a 2D network and a 3D network, then shifting of data can be simplified and more efficient. The 512 PEs are organized as an 8x8x8 cube and an 8x64 grid, such that shifting of the data can also be performed along the cube or along the grid. For example, a shift right of 74 PEs can be performed two ways: one way is to shift right 74 times along the 1D network, and the other way would be to shift up once along the cube, once north along the grid and two times right along the 1D network. Therefore a shift that would have taken 74 shift cycles can be reduced to 4 shift cycles using a dynamically reconfigurable multi-dimensional network.

3.2.2 Architecture of a Multi-dimensional Network

A reconfigurable multi-dimensional network can be dynamically reconfigured into different inter-processor communication network types. The ability to transform a linear array of processors into a cube of processors can be useful when the size of a shift is equivalent to the size of one plane of the cube. In this way, an entire shift can be done by shifting entire planes of data up or down in the cube, rather than shifting N PEs left or right. The shifting of data around the array can be done more efficiently and reduces the amount of computational time used for shifting. This decreases the communication bottlenecks, which enhances the efficiency of the computation.

In [14], the dynamically reconfigurable interconnect required that the SRAM cells be loaded with the configuration for the interconnect. This requires additional loading time and would increase the time to reconfigure the interconnect network dynamically. The reconfigurable interconnect of [14] was also used to reconfigure the network to bypass PE failures. For this design, the dynamically reconfigurable interconnect is focused on enhancing the PE communication by dynamically altering the interconnection configuration. Since C•RAM's communication is pattern based, the interconnect will be globally assigned as opposed to being assigned individually to each PE. By globally assigning the interconnection it is possible to change the configuration of the network quickly and efficiently, since there is no need to set up SRAM cells that control inter-processor connections.

3.2.3 Architecture of the 3D Network

For this research, the concept of creating a three-dimensional network to allow PEs to communicate in 3D was the primary focus. It is possible, however, to create many communication networks within a three-dimensional mesh network. As discussed in Chapter 2, many of the two-dimensional grid networks implemented toroidal, torus and spiral edge interconnection. Since previous array processors implemented these interconnection networks, the same interconnection networks are implemented in the three-dimensional processor mesh and extended to the third dimension of the mesh.

For the three-dimensional network, a three-dimensional toroid, where all dimensions perform shifting in a torus nature, is created. This toroid can then be broken down into its basic components, where each dimension of the mesh can perform torus operations independently. In this way, the network can be reconfigured to allow for torus shifting in one or more of the dimensions at any point in time, or to have no torus shifting at all. There are also other methods of shifting that can be added to the three-dimensional network, that can allow for the ability to create one-dimensional or two-dimensional shift networks. These network modifications are discussed in the next sections.

The development of a three-dimensional mesh network for C•RAM is proposed due to the introduction of the 3D SOI technology, where multiple dies can be stacked to create dense circuit designs. The SOI technology allows for a reduction in the amount of wiring per die by allowing some of the wiring to be done between the dies, using a via to connect two similar dies together to form a scalable design. The architectural concepts discussed here are independent of the technology that is being used; however, the design viability or density may be affected by the type of transistor technology used.

3.2.4 A 1D Network in a 3D Network

For previous C•RAM generations, a linear network was implemented, so it would be wise to continue to embed this 1D network in any new network. A three-dimensional network is composed of the same PEs as a one-dimensional network; this implies that the three-dimensional network can be expanded to include the one-dimensional network. The three-dimensional mesh can be transformed into a one-dimensional mesh by taking the first processor at position (0,0,0) and the last processor at position (N,N,N), and looking at the processors in the middle as points on a string. Each processor is given a processor ID number that would be associated with a point on the string and a point in the mesh. In Figure 3.2, one possible PE ordering can be seen where a one-dimensional network is created within a three-dimensional network. This requires the addition of a spiral left-right shift network. This spiral network can be used at any time, but is designed more specifically for the one-dimensional network.

The spiral network can also be opened or closed, so that PE 26 can be connected to PE 0, resulting in a closed spiral (torus) for the one-dimensional communication network. The option of a closed spiral is implemented since previous

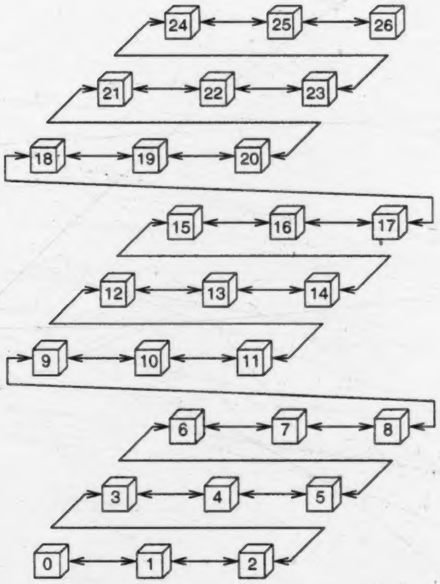


Figure 3.2: C•RAM - One-dimensional Interconnect in Three-dimensional Mesh

array processors containing a spiral network had this option. This one-dimensional communication network allows the PE array to operate in the same way as previous C•RAM architectural chips have operated. All PEs can communicate through a patterned left-right shift network to maintain a backward-compatibility with previous program coding.

3.2.5 A 2D Network in a 3D Network

When designing the two-dimensional interconnect network, it should be noted that two methods of creating the two-dimensional grid from the three-dimensional mesh were investigated. Unfolding the mesh into a rectangle of $N \times N^2$ is one method used to create a two-dimensional grid, and unfolding the mesh into a square of $N^2/2 \times N^2/2$ is another method to create a two-dimensional grid. Unfolding the mesh into a square is more difficult since it requires a much more sophisticated interconnect to be developed. The next architecture question is how to take advantage of the available architectural pieces. If the grid is formed using a square, then the one-dimensional network is unusable in the grid and would have to be redesigned.

Therefore, two spiral networks would need to exist for spiral shifting if the grid is unfolded into a square. If the grid is formed using a rectangle, then the fact that the one-dimensional network already works through the processors in an incrementing fashion can be reclaimed by unfolding the array in one of the ways seen in Figure 3.3.

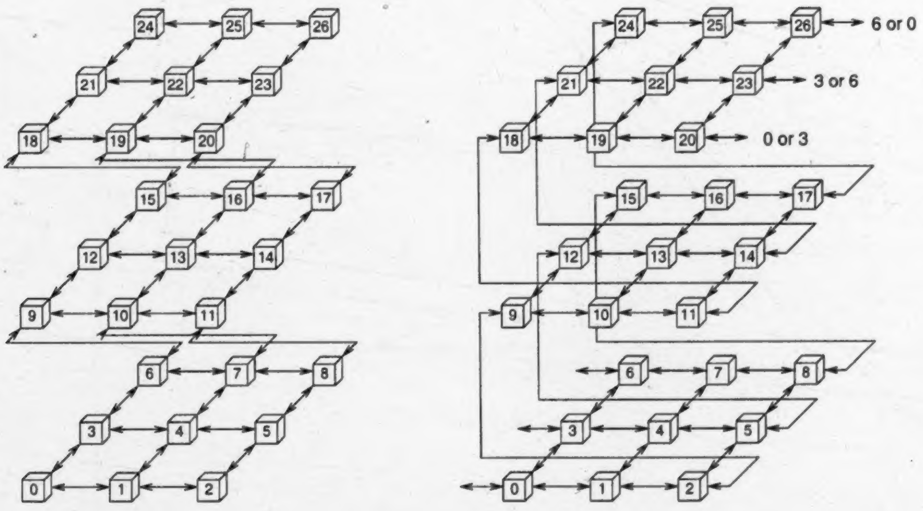


Figure 3.3: C•RAM - Two-dimensional Interconnect in Three-dimensional Mesh

Once the mesh has been folded out into a two-dimensional grid, it can then be modified to incorporate the different types of two-dimensional grid networks. The common two-dimensional grid communication network is the North-East-West-South (NEWS) network, where each processor can communicate to its four nearest neighbours. The three-dimensional mesh already has the base communication of the two-dimensional grid within each plane; therefore, the NEWS network is nearly complete. The addition of circuitry to produce a NEWS network across the planes is all that is needed to create a two-dimensional grid out of a three-dimensional mesh. There are also other modifications that need to be added to the NEWS network: torus across all directions of communication, torus across only one direction of communication, no torus communication, spiral communication for one direction with or without torus communication in the other direction, and open or closed spiral communication.

3.2.6 Multiple Chip Interconnection

It has been mentioned that previous C•RAM designs have had the capability of scaling the processing capabilities by extending the interconnection over multiple chips. This allows for the shifting of data from chip to chip, thereby maintaining the flow of data and increasing the amount of processing power available. It was also mentioned that this method of scaling the processing power should be maintained; however, the implementation of this scaling method raises the problem of the number of pins needed to maintain full interconnection between dies. For example, a 4x8x4 (x,y,z) cube would require 64 bi-directional pins for the left-right interconnect, 64 bi-directional pins for the north-south interconnect, and 64 bi-directional pins for the up-down interconnect. This means that 192 bi-directional pins are required for shifting data. If we take the same block and replicate it on a die such that each die contains an 8x8x4 (x,y,z) cube, and then two dies are stacked together using the MIT LL 3D SOI technology, the number of pins required for shifting becomes 384. The problem of increasing the number of pins on the die results in the die area being more pad-limited than core area limited.

To solve the problem of an ever increasing pin count, it was determined that the shifting of data off-chip should be multiplexed. By multiplexing the shift data, the number of pins required for off-chip shifting is reduced to a reasonable number of pins. The number of pins to use for shifting is determined from the size of the cube that is implemented in a single package. This multiplexed shifting circuit takes into consideration the possibility of plane failures that can interfere with the left-right and north-south interconnection shifting.

This off-chip shifting circuit is able to operate properly under all of the configurations of the communication network. In other words, the shifting of data during 1D communication should only depend on one bit of the shift pins, since in 1D communication only one PE is sending data off-chip and only one PE is receiving data from off-chip. The off-chip shifting controller is designed to take care of all the shifting conditions and the resultant behaviour. This controller is designed to respond to a shift clock signal that controls the shifting of data between chips or for a single chip to external circuitry.

In Figure 3.4, the shifting is simplified to a 2D array. In this figure we see how the number of pads required for off-chip shifting is reduced through multiplexing the data. This method of multiplexing can be extended to a 3D cube in much the same way as it is shown in the figure. A shift controller that automatically increments an internal address that corresponds to the multiplexer port connected to the pads is designed in the next chapter. The use of a shift clock signal that is common to all chips in the design allows for the shifting of data to be synchronized, such that all chips are shifting the appropriate data in and out.

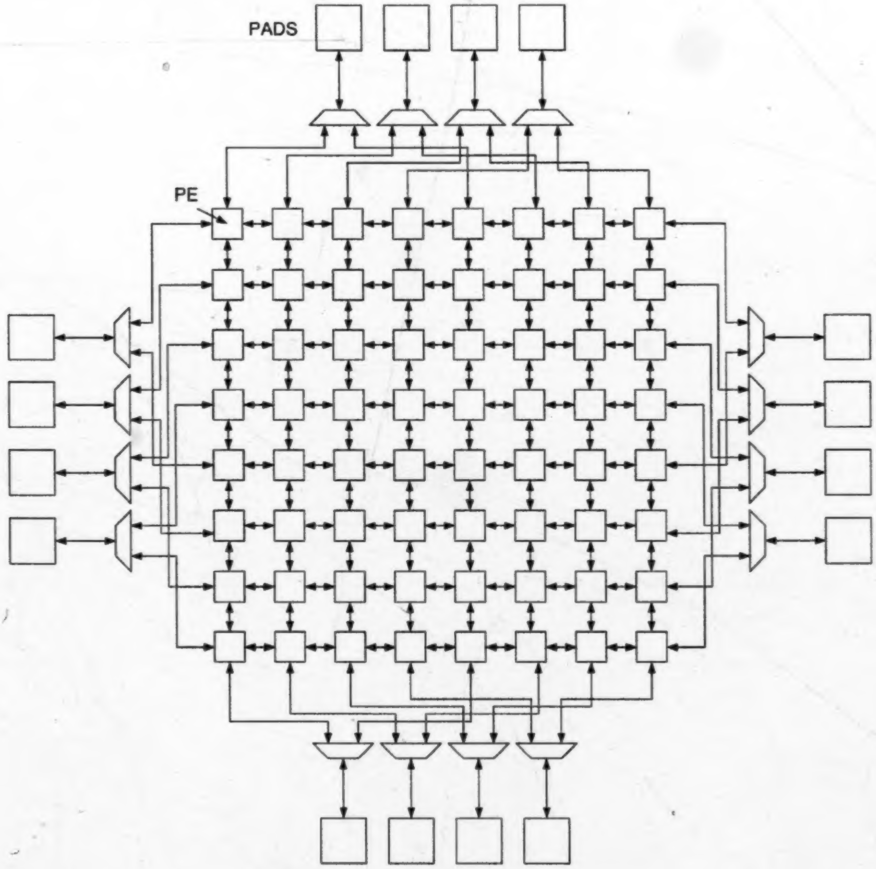


Figure 3.4: Off-chip Data Shifting

In the development of a test chip it is advantageous to have additional pins to perform shifting, since most academic testers do not have bi-directional test pins. For this reason, a fixed number of input pins and a fixed number of output pins are added along with a pin for identifying tester operation. In this way, the shift controller and shifting circuits could be easily tested on a tester that does not have bi-directional test pins.

It should be noted that a problem becomes apparent when looking at multi-chip interconnection as it relates to the 1D multi-chip communication network. It is most evident when looking at the spiral across multiple chips, since the spiral is designed specifically for within a chip package. When shifting using the 2D and 3D configurations, data is communicated between chips, and data PE adjacency ordering is maintained. In the 1D multi-chip configuration, the PE adjacency is not maintained relative to the 2D and 3D configurations; thus, the dynamically reconfigurable nature of a multi-chip system may not be feasible.

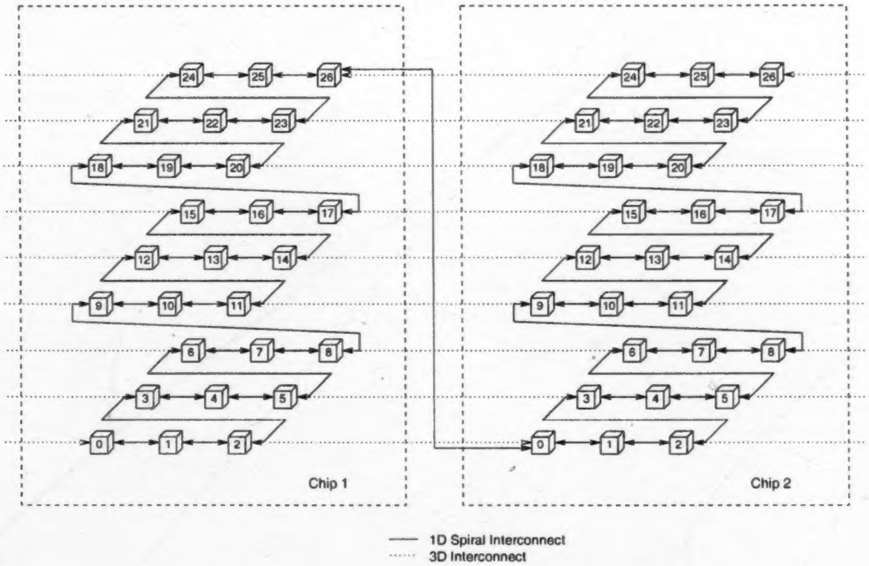

24
25
26
21
22
23
18
19
20
15
16
17
12
13
14
9
10
11
6
7
8
3
4
5
0
1
2
Chip 1
24
25
26
21
22
23
18
19
20
15
16
17
12
13
14
9
10
11
6
7
8
3
4
5
0
1
2
Chip 2
1D Spiral Interconnect
3D Interconnect

Figure 3.5: Multi-chip 1D and 3D Problem

3.3 Redundancy Design

Redundancy has been a common design addition when working with memories because of the high density of memory cells and higher possibilities of manufacturing defects affecting the cells in a column or row of memory. When creating redundant elements it is better if the redundant element is identical in every respect to the element that it is replacing. For memory, if a redundant memory element is different from the basic memory element, the performance or reliability of the redundant memory cells can alter the functionality of the design. If the performance or reliability is worse than the basic memory element, the total speed of the design and its reliability will be compromised. It is also possible that the redundant element is more reliable and faster than the basic element; in this instance, there should be no compromises in the speed of the design or its reliability. For memories, additional rows and/or columns of memory are added to increase the chances that the final die is fully functional. The addition of redundant rows and/or columns of memory may be needed to increase the yield of C•RAM, since a large portion of C•RAM is memory. Since C•RAM's aim is to attach a single PE to every column of memory, every additional (redundant) column of memory should result in a redundant PE. Assuming that the defect density can be anticipated, the cost of redundancy on the yield can be determined. With the cost and yield improvement, it will be possible to determine the cost effectiveness of the redundancy. As this is an academic design, it has been decided that the actual design will contain all the PE redundancy designs to improve the chances of obtaining a fully functional test chip.

3.3.1 Compatibility with Memory Redundancy

C•RAM is a processor-in-memory variant, which can benefit from the addition of redundancy since a large portion of the design is memory. The redundancy mechanism for failed rows can be identical to that of standard memory arrays; however, the mechanism for failed columns needs to be different since the memory is identified by both the column address and the PE adjacency. For C•RAM, the basic column redundancy is performed by using PE redundancy to skip the failed column of memory, as seen in Figure 3.6 and Figure 2.4. For example, when a column of memory is found to be defective, the column of memory and the PE associated with the column of memory are flagged as defective and then skipped. The PE and column are replaced by a redundant PE/column element. This also happens if the PE itself is found to be defective. This technique is very limited when the memory column is bad but the PE is good, because then the good PE is discarded. Additionally, if the column of memory is good but the PE is bad, then the good column of memory is omitted as well. This can result in a non-functional chip if the two scenarios occur within the same redundancy block. Defects are not uniform in size or uniformly distributed over the die; therefore, a single defect can cause one or more columns or PEs in a group to be faulty. It is for this reason that additional levels of redundancy are added to the PE mesh. These levels of redundancy will be discussed in the next section. Two additional memory redundancy options for performing column redundancy in C•RAM are designed.

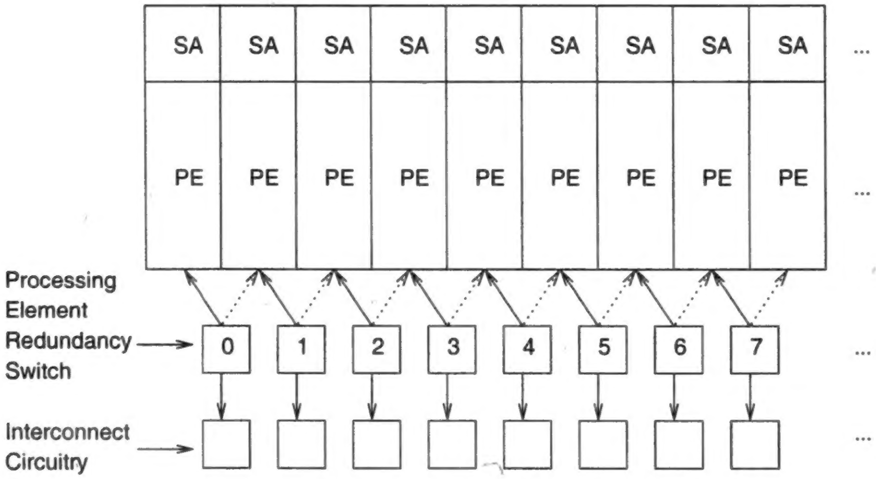


Figure 3.6: C•RAM Redundancy of Processing Elements and Memory (original)

The first option involves the addition of a switch that allows a single PE access to two possible columns depending on which column has failed. In Figure 3.7, the architecture can be seen where one PE is connected to two (or more) columns of memory through a redundancy switch. This technique is also limited in that the addition of an extra column of memory will be needed to maintain the ability of all PEs to access good memory. This option is limited since the aim of C•RAM is to attach every column of memory to PE, and it can be seen that not every column of memory will be directly connected to a PE. In this option, every ten columns or groups of columns will be connected to only nine PEs. Due to this limitation, the second option was formed.

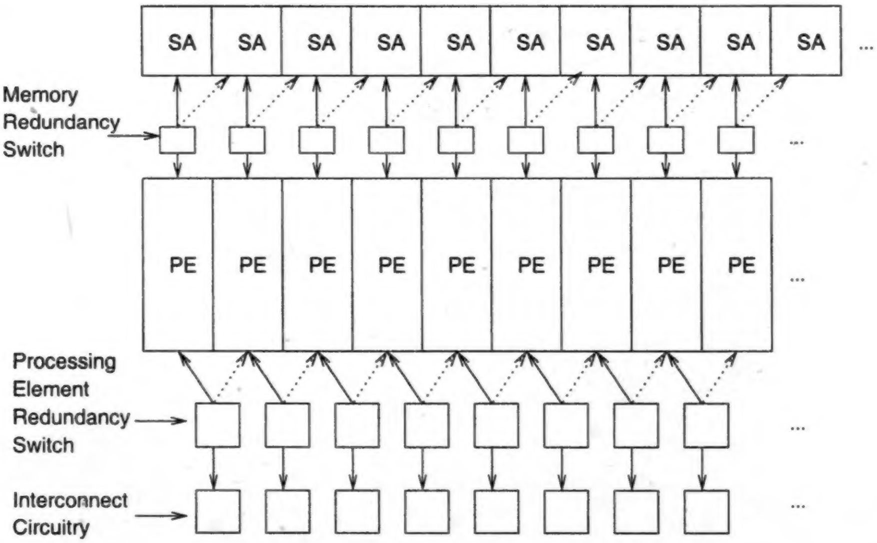


Figure 3.7: C•RAM Redundancy of Processing Elements and Memory (option 1)

The second option, in Figure 3.8, involves the addition of a switch that combines the effects of memory redundancy and PE redundancy. This way the memory columns are still matched to the PEs, and the bad PE or bad column of memory can be bypassed independently. The possibility of failure is reduced even farther than before. The simplified version of this option is seen in Figure 3.6, where the switch between the sense amplifiers and the PEs is removed. By simplifying this option, the PE redundancy switch would be used to perform redundancy for the memory and the PEs. This option is advantageous since each column of memory is connected to a PE, thereby meeting the column requirement.

In both options, the memory redundancy switch requires the result of a register

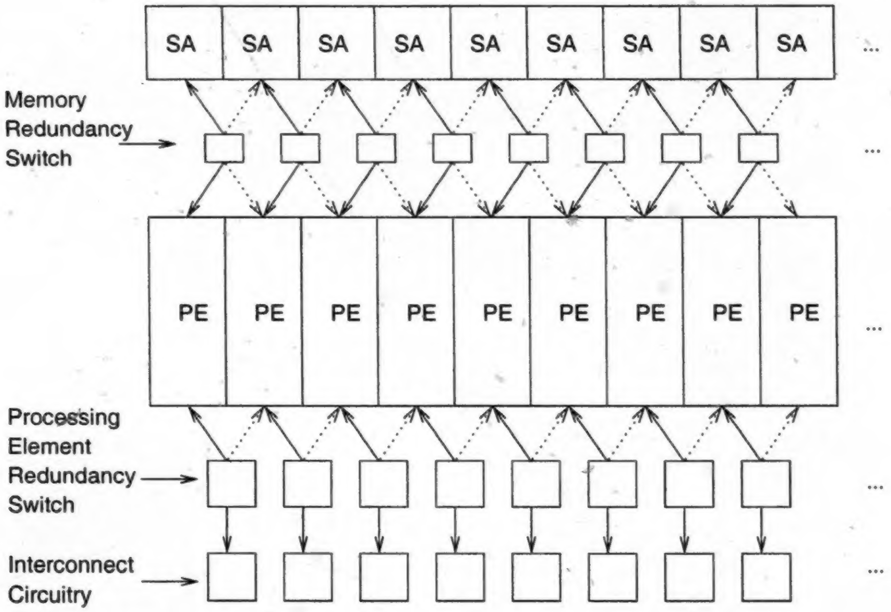
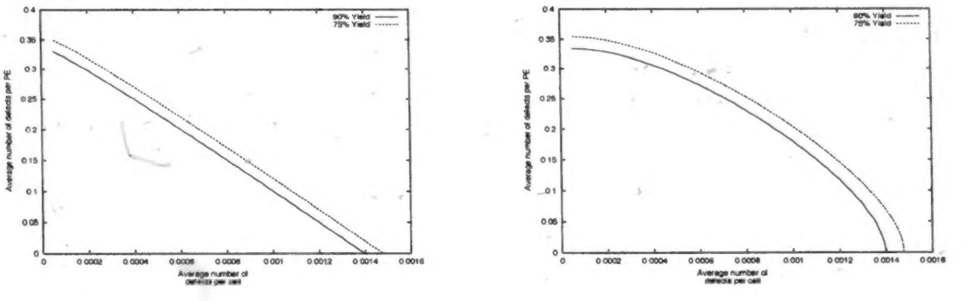


Figure 3.8: C•RAM Redundancy of Processing Elements and Memory (option 2)

that stores the skip result, such that if the skip register for the sense amplifier is set, then the switch is flipped to the next sense amplifier. The skip register can be placed within the switch or within the sense amplifier. Both options also are able to maintain memory addressing so that all column addresses will address a valid column of memory. This is done by incorporating the column enable signal into the redundancy mechanism, such that the column enable signal traverses through every redundancy switch before finally reaching the sense amplifier databus connection switch. Therefore, it will still be possible to identify each PE by a column address, though its physical location may not correspond to its logical location. The column redundancy mechanism will maintain the column address and PE adjacency requirements. If a failure occurs in the memory redundancy switch, it will result in the complete failure of the chip. When determining the final yield of the design, the possibility of a failure in this area must be considered as well.

The first option is not feasible, since it does not meet the requirement of attaching a PE to each column or group of columns of memory. To determine the value of the second option versus the original, the yield and cost need to be investigated.



(a) Column Redundancy using PE Redundancy (original)

(b) Independent Column and PE Redundancy (option 2)

Figure 3.9: Yield Plots for Memory and PE redundancy

In Figure 3.9, the yield graphs for a 9x9x9 PE array mesh with 260 rows of memory are shown. The graphs show only the yield as it relates to the defect density that it can accommodate, showing the original, an implementation of memory redundancy through the PE redundancy, and the second option, in which the memory redundancy and PE redundancy are split. The graphs contain the plots for 90% yield and 75% yield. From the graphs, we can see that the separation of memory and PE redundancy results in a better overall response to the defect density of the technology. Though the graphs indicate that the separation of memory and PE redundancy will result in better yields, the additional area cost of this redundancy should still be considered before incorporating this redundancy mechanism into the design. In Section 4.8.3, the yield as related to area cost is shown.

In [16], the PE redundancy mechanism was also used to perform PE and column redundancy. The 3DSOI C•RAM, however, did not include column address redundancy. The mechanism was able to bypass PEs and columns of memory for shifting operations, but the memory addresses for the failed column or PE were still addressable. Therefore, it was still possible to access invalid data, and the column address and PE adjacency were not consistent. The means of bypassing failed PEs and columns of memory is still maintained in this design, and the addition of column address redundancy allows uninterrupted addressing of PEs and memory. In 3DSOI

C•RAM, the user or programmer must include test and setup configuration into the programming code, as well as code that reconfigures the addressing of memory. In this design, no reconfiguration of memory addressing should be required, as the the column addressing is incorporated into the redundancy. The need for initial test and setup is still required to initialize the C•RAM array to avoid faulty PEs and columns of memory. With 3DSOI C•RAM, it is possible to use all the PEs and columns of memory on the chip if no failures are found; this is, however, not possible in this design.

3.3.2 Processing Element Redundancy

The redundancy of the processing elements is needed, since it is the next most likely failure point in the chain based on area and density. By adding redundancy for the processing elements, the possibility of a chip failing as a result of a faulty PE is reduced. By adding redundancy to the design it is possible to increase the total yield for the design, thereby offsetting the design cost of adding redundancy. When looking at processing element redundancy in C•RAM, two options arise: one required the addition of a processing element that was slightly different from the non-redundant PE, and the other required the addition of an identical processing element. The first method, where a modified processing element is used, was designed in such a way that when one PE failed, it connected itself to a redundant bus. This redundant bus passed the failed PE's connection points to the redundant PE so that the redundant PE will in effect appear as if it was located in the failed PE's position. If the redundant PE is not identical to the other PEs, more design work is required, and it may result in additional failures. The redundant bus will result in slower shifting for the PE that is connected to the redundant bus, since all the PEs in the group have their redundant signal switches attached to the redundant bus. Therefore, the redundant bus has increased capacitance due to the switches. Total operation will also be slower since the signals must be passed from the failed PE to the redundant PE. This method is limited in that only one PE in the group can fail. If more than one PE in the group fails, then the whole chip is non-functional.

This can, however, be fixed by creating a grid layout of redundancy using buses. This is seen in Figure 3.10; however, this requires that two S registers be available in order to connect the faulty PES to the proper redundant bus.

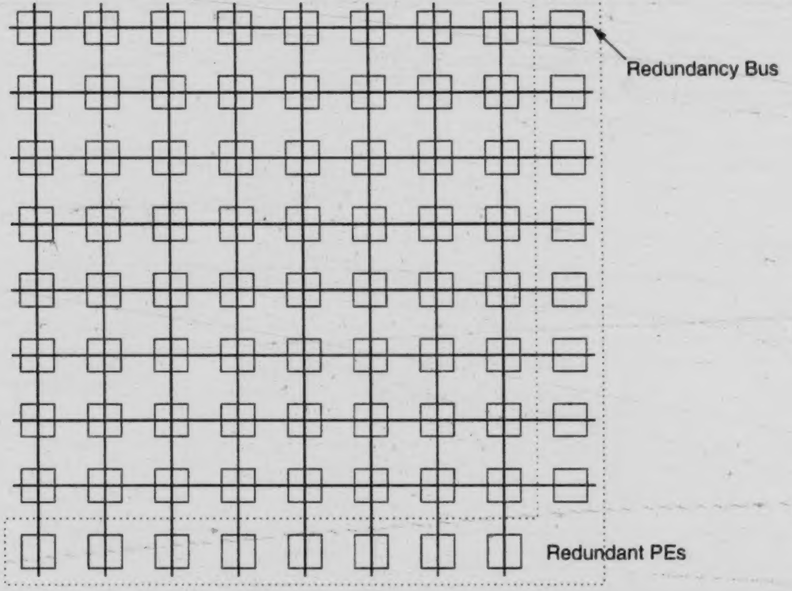


Figure 3.10: C•RAM Grid Redundancy using Redundant Bus

The second option was produced when the embedded memory column redundancy was discovered. The embedded memory redundancy scheme was aimed at moving the problem down the array. The second option, where all the PEs are identical, can be seen in Figure 3.11. This option allows for the replacement of the PEs by a switch network, thereby allowing all PEs to have similar delay paths, as well as identical design. This results in less design work being needed for the PE, allowing for more design work to be performed on the interconnection network and interconnection redundancy. In this option it is also possible to see that it can only accommodate one PE failing in the group; however, this option can easily be extended to a two-dimensional grid of PEs where the two-dimensional grid is redundant in two-dimensions.

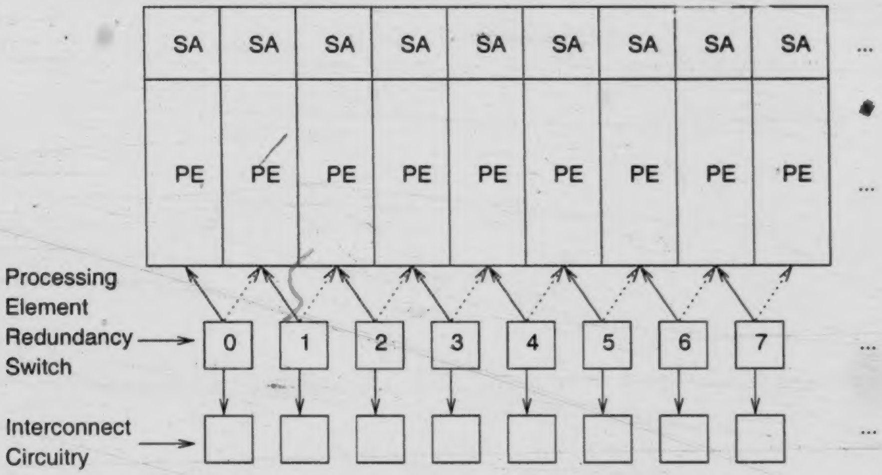


Figure 3.11: C•RAM Redundancy of Processing Elements

3.3.2.1 Grid Redundancy for PEs

When looking to create redundancy on a two-dimensional grid, previous implementations of grid redundancy are sufficient. For the proposed fault-tolerant three-dimensional interconnection network to be possible, the two-dimensional redundancy must maintain the interconnection between planes. Previous implementations of grid redundancy were aimed at embedding a fault-free grid within a faulty grid. This method, however, removes the ability of maintaining the interconnection between the planes. The proposed three-dimensional interconnection network requires that all planes in the cube have a uniform fault-free grid; this is not possible using current grid redundancy techniques. Therefore, a new redundancy grid that maintains a uniform plane for use in a three-dimensional cube is proposed.

In order to build a fully redundant PE grid, it is necessary to have the ability to perform redundancy in more than one dimension. This is done by creating a two-dimensional grid of PEs, and allowing there to be at least one additional PE for each dimension. For example, an 8x8 PE grid would be built from a 9x9 PE grid to allow for redundancy. For simplicity, we call the column of PEs X and the row of PEs Y. If there is more than one failure in the X direction, then one of the PEs needs to be replaced by one of the redundant PEs in Y, and the other PE can be replaced by the redundant PE in X. In Figure 3.12, we see how PE failures can be replaced using the redundant PEs in both the X and Y directions. In this way, the ability to replace the failed PEs is enhanced by extending the redundancy into another dimension. Four failed PEs are easily replaced using this method of redundancy, although only three out of the four PEs is in the area that needs to be repaired. From this small example, it is derived that the method of replacement should work properly. When moving to a large sized two-dimensional grid, the addition of one PE per dimension remains. In this design, the two-dimensional plane is required to be 8x8; therefore, the actual grid is made of 9x9 PEs. This corresponds to 17 redundant PEs for every 64PEs; about one-quarter of the array is made up of redundant elements.

This can improve the yield, since additional PEs are available for repairing or replacing non-functional PEs or columns. In this, way two failed PEs within a group of PEs can be replaced by a redundant PE from the second dimension of redundancy, as seen in Figure 3.12. It can be seen that a fault-free 3x3 grid is embedded in a faulty 4x4 grid of PEs. This is the aim of this research: to create a fault-free mesh within a faulty mesh. For example, an 8x8x8 fault-free mesh can be embedded in a 9x9x9 mesh that contains faulty PEs, assuming that the number of faults or defects does not exceed the number of redundant PEs available for replacement. If an 8x8x8 fault-free mesh is not possible, the device should be discarded. If, however, there exists a number of fault-free 8x8 planes less than 8, this device can be used as a diminished device due to the addressing redundancy and plane interconnect redundancy mechanisms.

It should be noted that a failure in the redundancy switches will result in a complete failure of the two-dimensional grid of PEs. For this reason, it was decided that an additional dimension of redundancy be incorporated. This will be termed as plane redundancy, where each plane is a two-dimensional grid of PEs that can be replaced by a redundant plane. The implementation of a global skip register for the planes that can be set externally in the case that the redundancy switches for the plane have failed is also added. This skip register will be OR-ed with the results of the redundancy switches, such that if the skip register is '1' or the redundancy

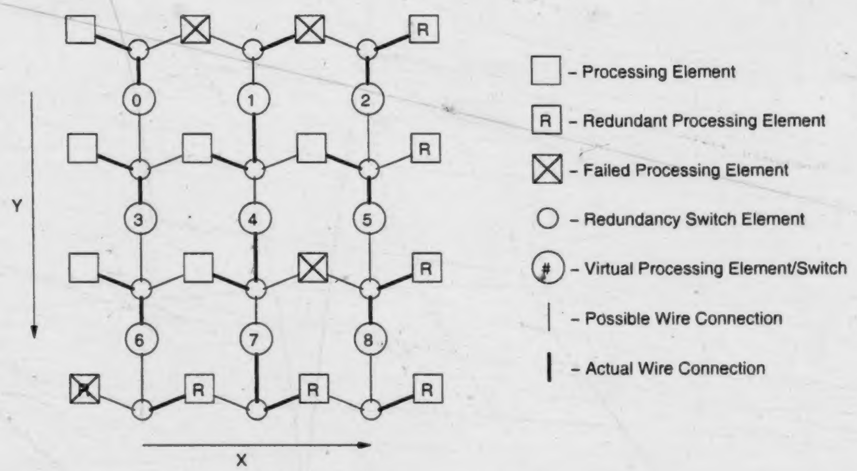

R
0
1
2
3
4
5
6
7
8
Y
X
– Processing Element
R – Redundant Processing Element
– Failed Processing Element
– Redundancy Switch Element
# – Virtual Processing Element/Switch
– Possible Wire Connection
– Actual Wire Connection

Figure 3.12: Two-dimensional PE Redundancy

switches result is '1', the plane will be skipped. To ensure that the plane is removed from all the appropriate communication networks, the skip result for the plane is sent back to all the PEs for the plane to be combined with each PE's local skip register.

Section 4.8.3, shows the yield improvement for each level of redundancy. In the section, the area cost for the redundancy mechanisms are seen. From the yield and cost, the best option of redundancy can be chosen that gives the best yield versus cost analysis.

3.3.2.2 Plane Redundancy

Plane redundancy is the redundancy mechanism for the z dimension of a three-dimensional mesh, where redundant planes of PEs are added to perform replacement when an entire plane of PEs fails. By adding entire planes of redundant PEs, the probability of a fully functional design is higher, even when a plane of PEs fails. The plane redundancy must allow for full addressing of the PE array, as well as maintaining the inter-processor connections for a three-dimensional mesh. When a plane fails, the plane must be bypassed, thereby allowing all data to flow past the failed plane to maintain the shifting of data. The building of plane redundancy should make allowances for addressing the good planes sequentially, even if there are failed planes.

For the PE redundancy, plane redundancy is performed through the interconnection network. The interconnection network is used to perform the third level of redundancy, because it is not feasible to use only some of the good PEs from a plane. Replacing planes that are found to be faulty with another entire plane ensures that the interconnection of PEs in the grid of the plane will maintain proper PE adjacency to other PEs on the grid, as well as being able to maintain PE adjacency between PEs on different planes.

3.3.2.3 Yield Improvement Through Redundancy

It has been discussed that redundancy is added to improve the die yield in order to obtain the highest number of working dies. In this section, the yields of each type of redundancy are examined in an attempt to determine the best redundancy mechanism. In Figure 3.13, the 90% yield plots for each redundancy mechanism are shown. For 1D redundancy, the switch method of redundancy and the bus method of redundancy will have the same yield, since both methods perform redundancy through the replacement of one PE per redundancy block. The same is said for the combination of 1D redundancy with plane redundancy (3D), where plane redundancy replaces entire planes. After these two methods of redundancy the results diverge, since the grid redundancy using bus replacement of PEs is not as flexible as the grid redundancy performed using grid switching. Plane redundancy (3D), as discussed previously, is based on replacement of an entire plane of PEs rather than replacing single fault PEs, as is done in the grid redundancy. This is to help in maintaining the grid interconnect.

In the graph we can see that the 1D and 2D bus redundancy results in better yield than the 1D and 3D redundancy mechanism. This is due to grid redundancy being more flexible, because 3D redundancy replaces an entire plane while 2D redundancy will only replace single faulty PEs. We also see that the 1D and 2D switch redundancy mechanism, in Section 3.3.2.1, results in better yield response than the bus redundancy mechanism containing 1D, 2D and 3D redundancy.

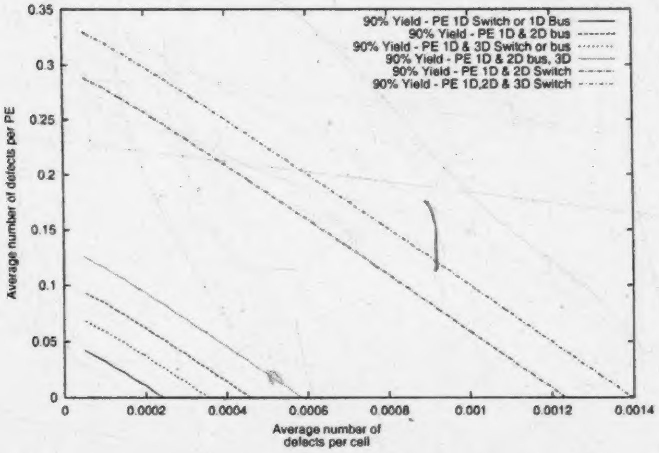


Figure 3.13: 90% Yield for PE Redundancy

Though the graphs indicate that the PE redundancy built, using switches to perform grid redundancy, will result in better yields, the additional area cost of this redundancy should still be considered before incorporating this redundancy mechanism into the design. In Section 4.8.3, the yield as related to area cost is shown.

3.3.2.4 Addressing Redundancy

When addressing memory in DRAMs or SRAMs in a complete chip, the address should always be pointing to something valid. In DRAMs or SRAMs with redundancy, this is done by using fuses to set failed addresses to point to new areas of the memory array. For C•RAM designs, however, this can be very difficult, since the PEs are attached to columns of memory and the position of the PE indicates its position in the interconnection scheme. If the same scheme as a DRAM or SRAM was used, then the interconnection scheme would have to be modified to accommodate the failure of a column of memory or PE. The PE redundancy is modified in such a way that the failed PEs can be replaced using a switch network. The interconnection of PEs then becomes independent of the failure of a PE, and is dependent on a separate switch for performing the interconnect. This scheme can be extended to the addressing of the memory and/or PE, so that the addressing of memory or PE is not based on the physical memory or PE location, but rather on the redundancy switch.

Therefore, the redundancy switch is how the PEs and memory are addressed. In Figure 3.11, the numbers represent the address used by the external memory interface. Both the redundancy switches and the interconnect switches have the same addresses.

This addressing redundancy can be extended further into the second and third dimensions. For the second dimension the addressing is similar to that seen in Figure 3.11, where the redundancy switch will extend to another row of PEs. The third dimension of addressing redundancy uses a different technique to maintain address continuity between the external and the internal world, so that the user does not have to create code to accommodate failures in planes of PEs. The technique used to maintain address continuity performs internal address recalculation for the planes. If a plane of PEs is found to have failed, then the internal address would be calculated using the external address. The failure signals that the next good plane is associated with the good external address, see Table 3.1.

Table 3.1: Plane Addressing Calculation on Failure

Physical Plane	Failure	Addressable Internal Address
0		0
1	X	None
2		1
3		2
4	X	None
5		3

3.3.3 Interconnection Redundancy

For the interconnection network, there is no redundancy of the interconnection switches for the PEs in a plane of the cube, since the probability of failure for the interconnect is minimal in comparison to that of the PEs or the memory array. The failure is minimal because the interconnection network has low density in comparison to the PEs and the memory array, since the interconnection network is composed almost entirely of wiring. Redundancy of the interconnection network is implemented between planes of the PE mesh, so that when a plane fails, it is bypassed. To allow one or more planes of PEs to fail, the interconnection network between the planes of PEs performs bypassing of the failed processor planes. To maintain full functionality, additional planes of processors would need to be added to the design for redundancy, and the ability to fully access every plane of processors would be useful. If the planes are not accessible and there are no failed planes, then the chip would be using power for unusable processors. It may also be useful to have the processors that are unusable disabled, along with the unusable switching circuitry that uses the most power, to minimize the power usage of the chip. In Figure 3.14, the interconnection redundancy is shown where the redundancy for the left-right interconnection is performed. The PE interconnection redundancy block contains circuitry for bypassing failed planes of memory while maintaining the left-right interconnection.

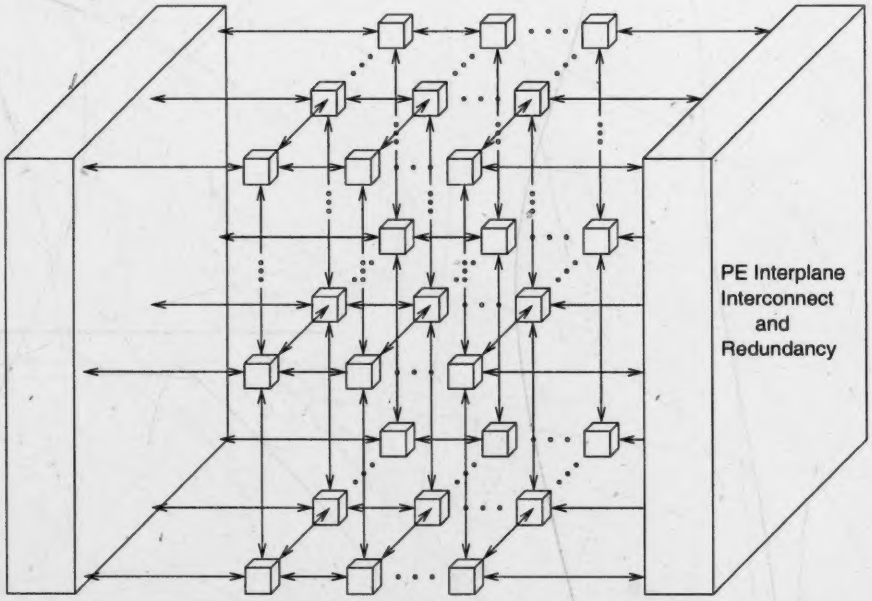


Figure 3.14: C•RAM Interconnection Redundancy

If one or more of the planes of PEs fails, the circuit needs to be able to maintain functionality by bypassing the failed planes and making it appear as if they do not exist. This way, all shifting operations can occur without the possibility of a failed PE shifting data to another PE. If the middle plane in Figure 3.14 has too many failures that cannot be repaired with the redundant PEs, then the data from the middle plane should not be passed to the next available good plane. Moreover, the data from the other good planes should not be used by the failed plane; the data should bypass the failed plane. This is also a requirement when shifting in the z dimension of the cube, so that the failed planes are bypassed entirely. The method of bypassing more than one failed processor plane is added, since it possible to use the MIT LL 3D SOI technology, which allows for the addition of extra dies to the stack to create a fully functional stack.

3.4 Summary

In this chapter, the modifications for the C•RAM architecture that allows for a dynamically reconfigurable and redundant PE array architecture are defined. The architectural changes allow for more complex parallelizable code to be run, since the interprocessor communication network is dynamically reconfigurable. The architectural changes to add multiple dimensions of redundancy allows for failures to occur in any of the three dimensions. It also includes the ability to repair or replace the failed PEs while maintaining all dimensions of communication with little or no delay penalty. All PEs would have a uniform delay for shifting between its 6 nearest neighbours. The addition of a separate memory bank for external access allows for computation and I/O to occur simultaneously, thereby reducing the amount of time stolen from computation. This allows for more computation to occur per second than was previously possible.

Chapter 4

Implementation

In this chapter, the architectural concepts proposed in the previous chapter are discussed in detail. This chapter details the design specifics. We chose the MIT Lincoln Labs $0.18\mu m$ SOI - 3D wafer stacking technology as the target process. The MIT LL process was chosen because of the ability to stack multiple dies to create a design that extends into the third dimension of space. This easily allows for the extension of C•RAM into a true three-dimensional mesh, which results in the routing per die being minimized. By sub-dividing the mesh over the dies, the routing of signals for each die is also sub-divided, and the wires to connect the pieces of the mesh are connected through the die.

4.1 MIT Lincoln Labs 0.18μm SOI

The base SOI technology at Lincoln Labs was added to by Burns et al. [3, 4], allowing for the stacking and connecting of multiple wafers (dies). This technology allows for the connecting of inter-die wires to adjacent dies in the stack. This introduces the ability to create readily scalable designs that depend only on the amount of scaling capability that was designed into the die. Similarly, a design can be segmented into multiple dies, where the wiring interconnect length between the dies of the design would be about $10\mu m$. Designs typically requiring that signals be routed $1mm$ or more across the design can benefit from the speed-up gained from a shorter wire. The design is beneficial, since the signal does not need to be buffered or boosted, and the propagation delay of the signal will decrease due to the reduced resistance and capacitance of the signal wire.

Due to the benefits of stacking, the creation of a three-dimensional interconnection network could take advantage of this technology. The ability to route interconnection wires vertically through the die to another identical die is beneficial when creating a three-dimensional cube of processors, since each die can contain one or more planes of the cube. The ability to stack two or more dies also allows for future expansion of the design. In addition, if the design is constructed properly to allow for redundancy, additional dies could be continually added until all of the chips are fully operational.

This SOI technology is also low voltage; it is designed to operate at 1.5V. This should result in a fairly low-power design when operating at high frequencies, as compared to a bulk-CMOS design operating at similar frequencies.

4.2 Memory Array

Our C•RAM memory array employs CMOS SRAM cells, which are lower density than DRAM. Since the technology process from MIT LL is not a DRAM process, SRAM cells are used as the memory elements, to be conservative. SRAM cells were used for the entire C•RAM memory array because of their robustness and low access times. As was mentioned in the previous chapter, the addition of multiple banks of memory per PE to C•RAM will be discussed in detail. The multiple bank architectural change requires additional columns of memory per PE. It also requires the addition of extra row decoders, as well as control circuitry to determine which columns of memory are to be connected to the PEs, and which column of memory is connected to the external databus.

4.2.1 Multi-bank Memory

The multi-bank memory in C•RAM can be constructed of two or more columns of memory per processing element. In the multi-bank memory, all the columns of memory are accessible through the databus and by the PE. The added enhancement is that all columns of memory can be accessed simultaneously with each of the banks (columns) of a PE accessing different rows. Four banks of memory were chosen for the design, since it was discussed that a four bank architecture would be advantageous for MPEG encoding. For the multi-bank design, it was decided that three of the four columns would be addressable by the PE supplying the two frames of data and a bank for storing the results, with the three columns accessing the same row simultaneously. This was done to minimize the number of row addresses that would be needed and to simplify the controlling of the sensing, precharging and addressing of columns. The fourth column of memory is connected to the external databus of the chip. In this way, the PE and external memory R/W can occur simultaneously.

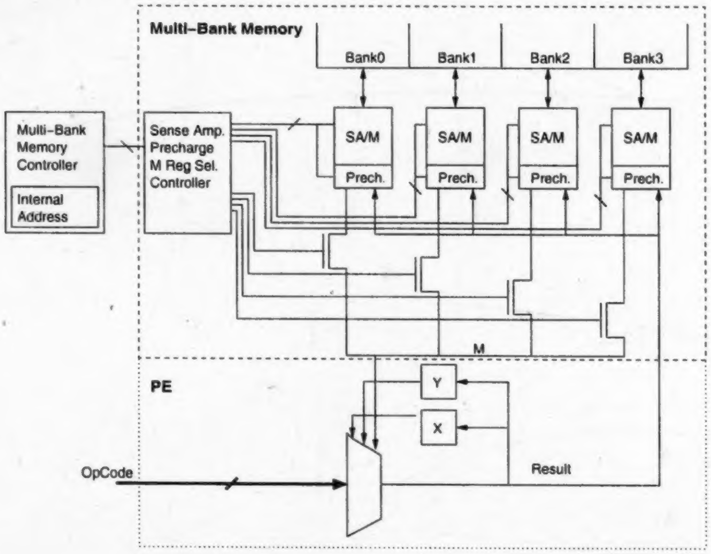

Multi-Bank Memory
Bank0
Bank1
Bank2
Bank3
Multi-Bank Memory Controller
Internal Address
Sense Amp. Precharge M Reg Sel. Controller
SA/M
Prech.
SA/M
Prech.
SA/M
Prech.
SA/M
Prech.
M
PE
Y
X
OpCode
Result

Figure 4.1: Multi-Bank Architecture with 4 Banks per PE

The simplification of the design allowed three banks to be simultaneously supplied with the same control data, which also simplifies the signal control for the PEs and the external memory bank. Additional control was added to select one of the three columns of memory to be the M register for the ALU and for performing reads/writes. The last aspect of the design requires that the column to databus connections be switchable, allowing the external I/O to connect to each column of memory. All of the above suggested that all columns be individually controllable to allow for individual column access as well as grouped column access.

Each bank of memory requires a sense amplifier in order to store the results of the row access or the result of the ALU operation. Each PE is assigned a sense amplifier group consisting of four sense amplifiers, with control signals being supplied by the multi-bank memory controller. The sense amplifier group has additional circuitry used to translate the external control signals and the multi-bank memory controller signals into bank specific control signals.

4.2.1.1 Word-line Decode

As was discussed, all columns are individually controllable. This means that all row decoders need to be individually addressable and controllable to allow for external I/O accesses and PE accesses to occur simultaneously. There are actually only two row control circuits need for this simplified design; one row control circuit is used to control the PEs' memory access, and the other row control circuit controls the external I/O row access. Only two row control circuits are needed since the design and control of the multi-bank are simplified. To perform row accesses, all the word-line drivers need to be able to retrieve the row signal from either the PE row address circuit or the external memory R/W row address circuit; see Figure 4.3. A selection circuit for each bank of memory, essentially, a multiplexer, which is controlled by additional circuitry, selects which row address or decoded row address to select depending on the internal address. Due to the way that the selection circuits are built, pre-decoding of the row address is still possible, so that a majority of the decoding of the address is done before the row enable signal arrives. This reduces the delay of the word-line enable signal once the row enable signal is activated. The internal address stores which bank of memory is currently being used for external memory R/W. The internal address is incremented each time that the bank-clock-enable is activated, indicating that the program is ready to progress to the next bank of memory. The controller circuit will be discussed later.

In Figure 4.4, an alternative method is presented where the row decoding is

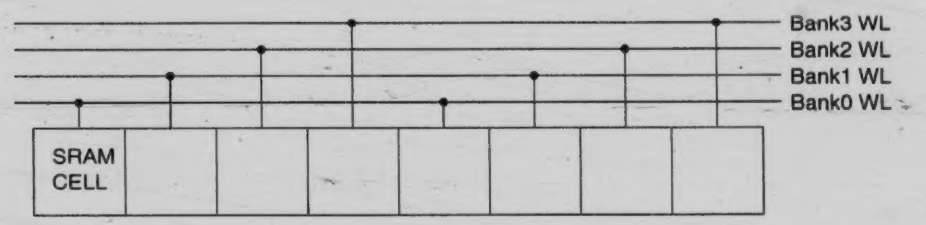

Bank3 WL
Bank2 WL
Bank1 WL
Bank0 WL
SRAM CELL

Figure 4.2: Word-line to SRAM cell

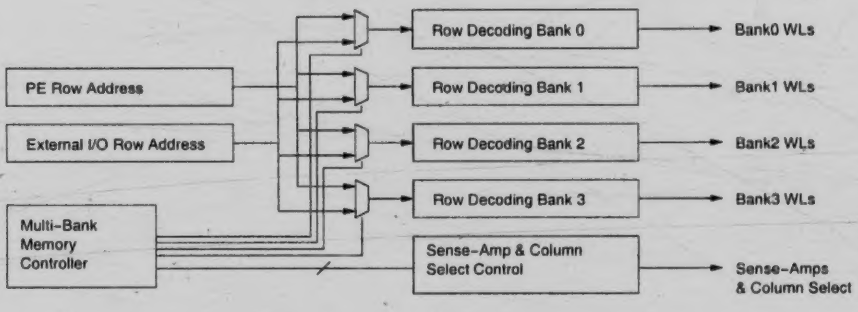

PE Row Address
External I/O Row Address
Multi-Bank Memory Controller
Row Decoding Bank 0
Row Decoding Bank 1
Row Decoding Bank 2
Row Decoding Bank 3
Sense-Amp & Column Select Control
Bank0 WLs
Bank1 WLs
Bank2 WLs
Bank3 WLs
Sense-Amps & Column Select

Figure 4.3: Word-line Decoding (option1)

done prior to multiplexer in attempt to reduce the number of row decoders that are calculating row addresses. This method requires that the proper row decoder outputs are supplied to the appropriate word-line drivers.

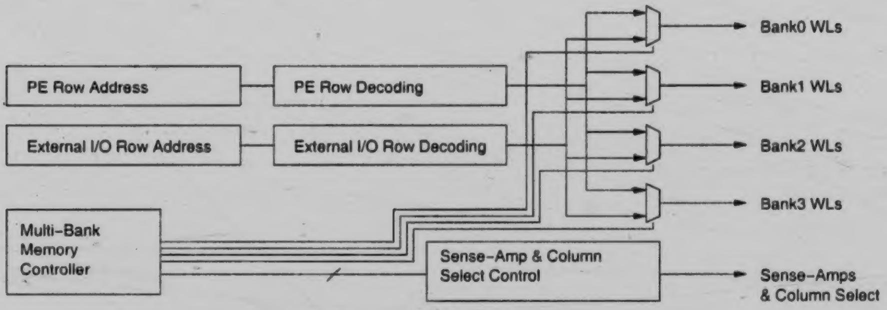

PE Row Address
PE Row Decoding
External I/O Row Address
External I/O Row Decoding
Multi-Bank Memory Controller
Sense-Amp & Column Select Control
Bank0 WLs
Bank1 WLs
Bank2 WLs
Bank3 WLs
Sense-Amps & Column Select

Figure 4.4: Word-line Decoding (option2)

There are advantages to both approaches, and certain costs associated with the advantages. For instance, if only two address are being used, as is the case in both figures, option 2 is likely to more advantageous since it will use less area, and the number of row decoders calculating addresses is reduced. This reduces the amount of power used to decode the row addresses, as well as occupying less physical area. This option, however, requires a selection circuit before each word-line driver; therefore, there are more selection circuits in option 2 as compared to option 1. This becomes more of a problem when the number of addresses used to access banks is increased, so that all banks are individually addressable. This option has the disadvantage of having a large number of selection circuits with more inputs, plus the row decoders for all the banks. In option 1, the advantage is that the design requires all banks to be individually addressable. Since the number of selection circuits does not change, only the number of inputs to the selection circuits change. Option 1, however, requires that all the row enable signals for the decoders go through a selection matrix as well.

For this design, option 2 was chosen due to the design choice of having only two row addresses: one for PE accesses and one for external accesses. This results in less area being used for row decoding, and less power as well.

4.2.1.2 Sense Amplifier and Bit-line Precharge

The multi-bank memory design requires changing how rows and columns are accessed or controlled, which also requires changing how the sense amplifiers and the precharge circuits are controlled. For multi-banking, each of the banks (columns) of memory of a PE must be individually controllable. Each bank contains a sense amplifier and precharge circuit, which is required to read and write the SRAM cells of the column. PEs can be built with a single column (bank) of memory or multiple columns of memory. When SA (sense amplifier) appears in a figure, it can be replaced by a single sense amplifier or a group of sense amplifiers (multi-bank). In Figure 4.5, the sense amplifier and precharge circuit for each bank of memory is shown. In the figure, all the necessary control signals and data are identified.

In order for C•RAM to work properly, a write-back circuit that is dependent on the WE register must be added. This is to ensure that only the PEs that have the WE register set to '1' should be overwriting the SRAM cell with the contents of the sense amplifier. This write-back circuit can also be used to speedup the overwriting

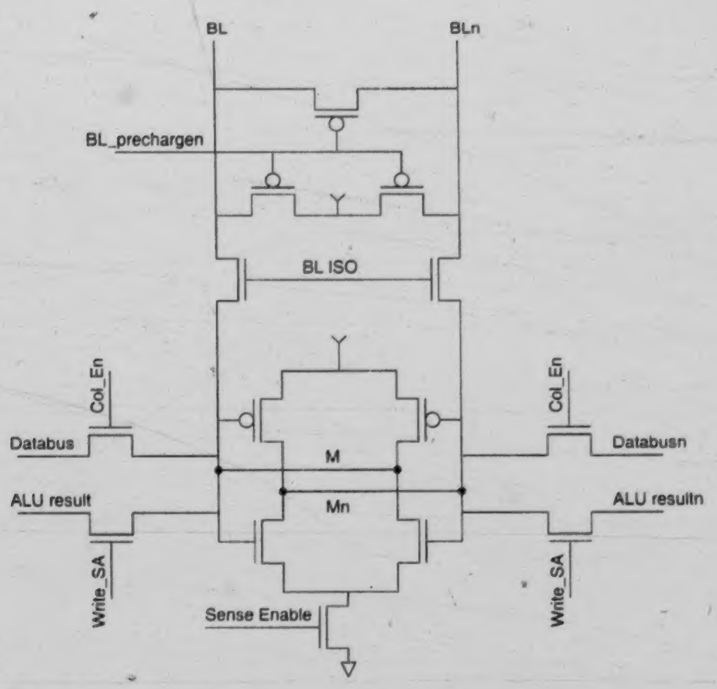


Figure 4.5: Sense Amplifier (M Register)

of cell contents by the databus. In Figure 4.6, a possible write-back circuit is shown where *M* and *Mn* are supplied by the sense amplifier, and *WriteBL* is generated. The *WriteBL* activates for external memory operations when the column enable signal for the column and the write memory signal are high, or for PE operations when the WE register is '1' and the write group signal is high.

Additional selection circuitry, which is similar to that used for the word-line decode, is used for the sense amplifier and precharge circuits to apply the control signals and connect data results of the PE. Since three columns are simultaneously available to the PE, three columns (banks) are controlled simultaneously to match the decoding used for the word-line decoding circuit. In this way, three of the banks have their data available at the same time, allowing for quick access to all three of the banks that are associated with the PE. A change in an external address, used to address one of the three sense amplifiers, will result in a new M register being available to the PE.

In Figure 4.7, the column controller can be seen where the multi-bank memory

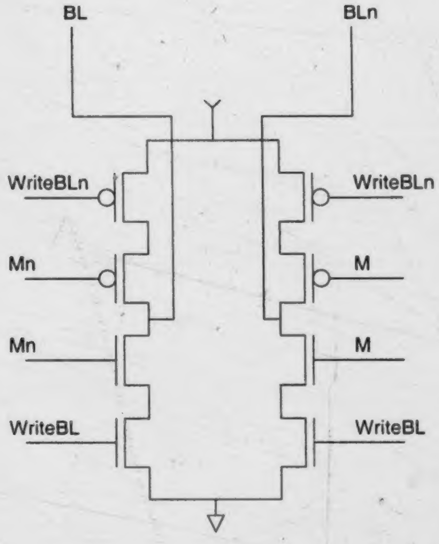

BL
BLn
WriteBLn
WriteBLn
Mn
M
Mn
M
WriteBL
WriteBL

Figure 4.6: Write-Back M Register to Cell

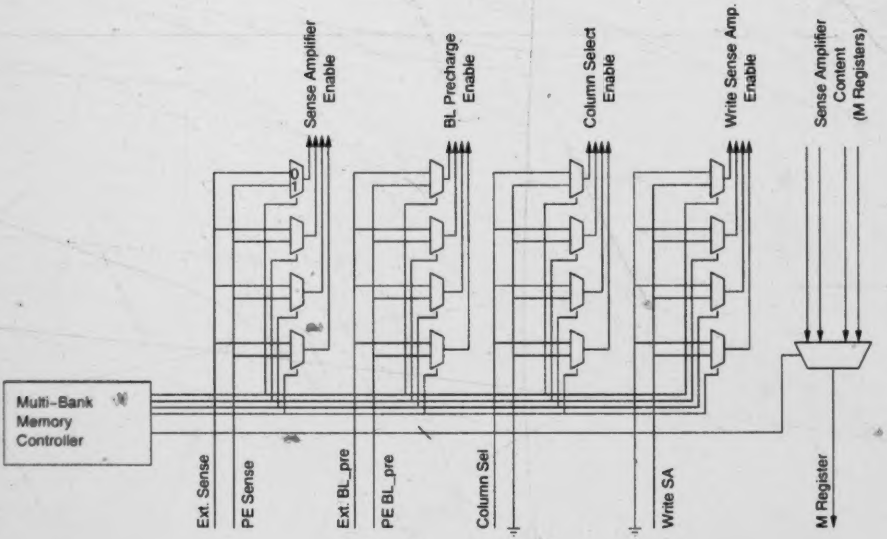

Sense Amplifier Enable
BL Precharge Enable
Column Select Enable
Write Sense Amp. Enable
Sense Amplifier Content (M Registers)
Multi-Bank Memory Controller
Ext. Sense
PE Sense
Ext. BL_pre
PE BL_pre
Column Sel
Write SA
M Register

Figure 4.7: Sense Amplifier, Precharge & Column Controller

controller is used in conjunction with control signals to control the bank signals. For instance, when the bank control signal is '0', the external signals will be sent through to the proper bank. In this way, each bank can be controlled individually or grouped together through means of the multi-bank memory controller. In Figure 4.7, the multi-bank memory controller sets the bank-enable signal to '0' for the selection multiplexer associated with the external I/O bank and to '1' for the selection multiplexers associated with the PE banks. For example, when the Ext. Sense signal makes transitions, the sense amplifier associated with the external bank will be enabled or disabled.

4.2.1.3 Column Decoding

For column decoding, only the external memory I/O address is necessary, so there is no need to have multiple column decoders. The column decoding signals are passed to the sense amplifier groups, and each sense amplifier group has circuitry for connecting the column (bank) of memory to the external databus. In Figure 4.7 and 4.8, we see that the databus is tied to the sense amplifiers through a switch. The switch is activated by the column decoding circuit and the column controller when the current sense amplifier group (columns of memory) is tied to the external databus. The internal address corresponds to the bank of memory that is attached to the external datapath.

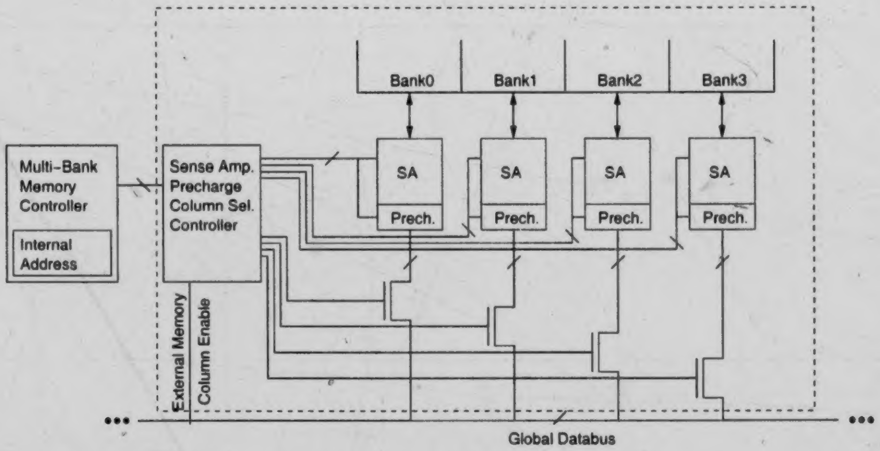

Bank0
Bank1
Bank2
Bank3
Multi-Bank Memory Controller
Internal Address
Sense Amp. Precharge Column Sel. Controller
SA
Prech.
SA
Prech.
SA
Prech.
SA
Prech.
External Memory
Column Enable
•••
•••
Global Databus

Figure 4.8: Column Decoding

In order to maintain the logical addressing and PE adjacency in the event of a PE or memory column failure, the column selection signals are propagated through all possible levels of redundancy before finally making their way to the sense amplifier group. In this way the failure of a column of memory or PE will not result in a column address addressing an invalid column of memory. In Figure 4.9, this method of column address redundancy is seen.

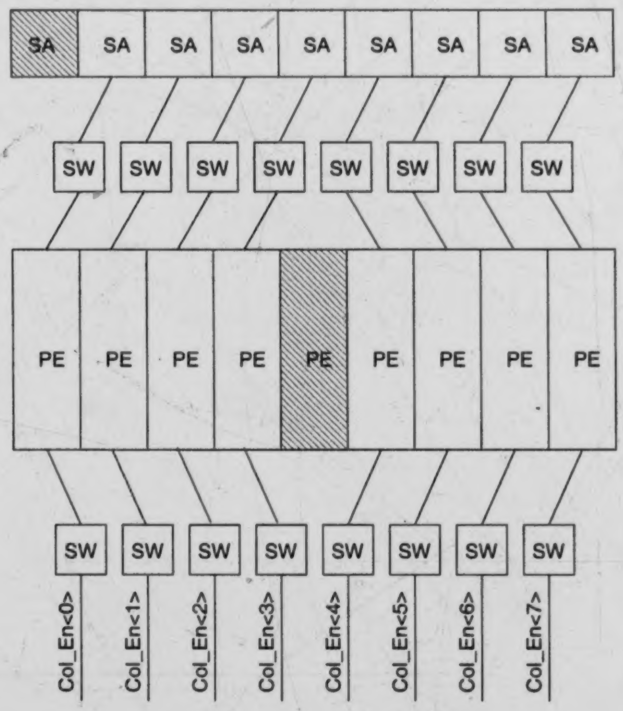

SA
SA
SA
SA
SA
SA
SA
SA
SA
SW
SW
SW
SW
SW
SW
SW
SW
PE
PE
PE
PE
PE
PE
PE
PE
PE
SW
SW
SW
SW
SW
SW
SW
SW
Col_En<0>
Col_En<1>
Col_En<2>
Col_En<3>
Col_En<4>
Col_En<5>
Col_En<6>
Col_En<7>

Figure 4.9: Redundancy of Column Decoding

For this design, the PEs are organized in a three-dimensional mesh such that each plane of the mesh has its PEs numbered from 0 to N. This corresponds to the column addressing being broken into two parts; one part is the plane number, and the second is the PE number on the plane. The combination of the two parts forms the actual column address. Therefore, since the column address is formed from the PE number on the plane and the plane number, the logical addressing can be further modified by plane redundancy. Thus, when a plane fails, its address is given to another plane. This allows the logical column address to always access a valid column of memory.

Plane Address Recalculation

The plane address recalculation is designed to allow the user to address all the planes of PEs that pass as being sequentially assigned. This means that if physical plane two fails, physical plane three is given the new plane address two. In Figures 4.10 and 4.11, we see two methods of address recalculation that re-evaluate the address of the planes in the three-dimensional mesh. Using recalculation, the user of the chip does not need to know which planes have failed, and the programs always address the same plane addresses.

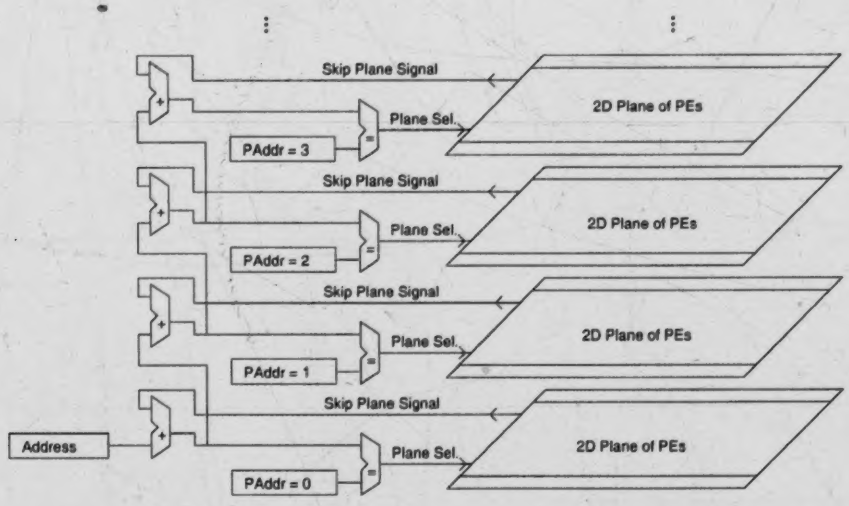

Skip Plane Signal
Plane Sel.
2D Plane of PEs
PAddr = 3
Skip Plane Signal
Plane Sel.
2D Plane of PEs
PAddr = 2
Skip Plane Signal
Plane Sel.
2D Plane of PEs
PAddr = 1
Skip Plane Signal
Address
Plane Sel.
2D Plane of PEs
PAddr = 0
+
=

Figure 4.10: Address Recalculation for Address Continuity

In Figure 4.10, the address is recalculated and compared against the internal plane address. Here, the address is recalculated every time the address changes. This means that there is a delay from the time that the address changes until all the planes receive the recalculated address. If the address matches, then the plane can be used to perform a column access, since the data is valid.

In Figure 4.11, the plane address is only recalculated whenever the contents of the S registers change. Since this usually only occurs at startup, the plane addresses are only calculated once as compared to the option where the address would need to be recalculated on every access. This method is superior in that the change of the

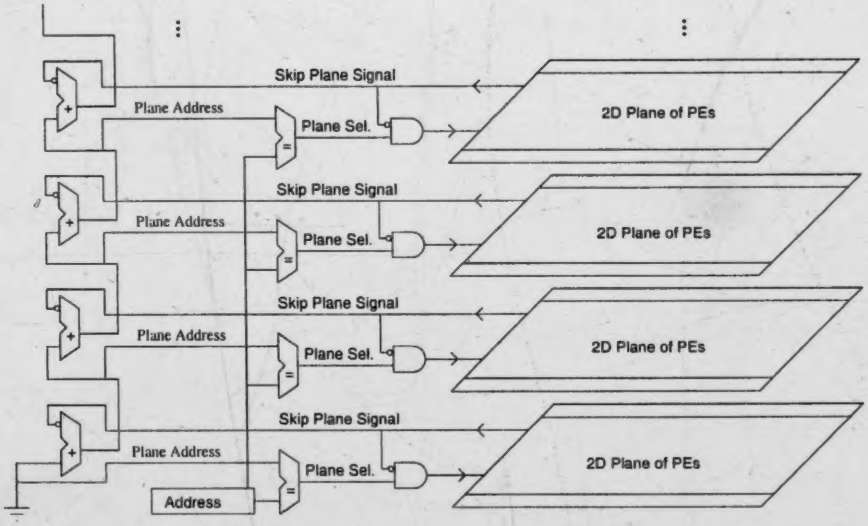

Skip Plane Signal
Plane Address
Plane Sel.
2D Plane of PEs
Skip Plane Signal
Plane Address
Plane Sel.
2D Plane of PEs
Skip Plane Signal
Plane Address
Plane Sel.
2D Plane of PEs
Skip Plane Signal
Plane Address
Plane Sel.
2D Plane of PEs
Address
+
=

Figure 4.11: Plane Address Recalculation for Address Continuity

external address only requires a comparison to enable the plane for external access. If the address matches and the plane is valid, then the plane can be used to perform a column access, since the data is valid.

4.2.2 Multi-bank Memory Controller

The multi-bank memory controller is responsible for creating the bank enabling and M register select signals. The multi-bank controller creates these signals to allow each bank to connect the appropriate control signals to the bank. Each bank is supplied with two signal groups; one group is used for external I/O memory R/W, and the other is used for PE memory R/W. Both signal groups contain a signal for controlling the activation of the sense amplifier, a signal for the precharge circuit, a signal for attaching the column to the databus, and a signal for overwriting the sense amplifier with the PE result. In Figure 4.7, the signal groups can be seen in detail.

In Table 4.1, the indexing of the M registers is shown. In the table, the M register address is used to address the sense amplifiers at each point in the cycle. The table also shows how the sense amplifiers are indexed, where ‘X’ represents the bank being accessed externally for loading or storing or data. For internal address 0, if the M register address being supplied externally is ‘1’, then the contents of sense amplifier 1 would be connected to the PE. If the M register address stays ‘1’ as it moves to internal address 1, the PE would switch its connection from sense amplifier 2 to sense amplifier 3. Similarly, as it moves from internal address 0 to 1, the sense amplifier that is reachable externally switches from 0 to 1. When the M register address is set to a valid address, the sense amplifier associated with the address will become the M register, and can be written to by the result bus of the ALU.

Table 4.1: Sense Amplifier Indexing

Internal Address	Sense Amplifier			
	0	1	2	3
0	X[1]	2	1	0
1	0	X	2	1
2	1	0	X	2
3	2	1	0	X
0	X	2	1	0

This can be extended to the MPEG encoding example. When M register address equals ‘2’, the results frame is connected to the sense amplifier; when it is ‘1’ the previous frame is connected; and when it is ‘0’ the current frame is connected. Starting at internal address 0, when the M register address is ‘2’, the sense amplifier 1 is connected to the PE. For as long as the internal address is 0, sense amplifier 0 can be connected to the databus. During internal address 0, it is assumed that all the possible memory cells connected to the sense amplifier 0s of the PEs are loaded with data for the next cycle, while the current cycle uses the memory cells of sense amplifiers 1 to 3 to perform computations. All data results of the current cycle are stored in memory cells of sense amplifier 1. As the cycle finishes, the internal address is incremented and data of sense amplifier 1 becomes the externally accessible. Now that the results of the previous cycle are accessible, they can be stored, and then new data can be loaded. The data that was previously loaded into sense amplifier 0 is accessible by the PEs, and is now addressable as the current frame (M register address '0'). Now that the previous frame data of the previous cycle is no longer needed, it is reasonable to overwrite the data with the results of the new cycle. This can continue indefinitely, since the internal address will loop over, as is also seen in the table.

---

[1] X indicates the sense amplifier that is connected to external memory and is unreachable using external M bank address

In Table 4.2, the data of Table 4.1 is extended to time-based addressing and access. At time 0, it is assumed that the internal address has been reset. In the table, M Register is a signal that is generated by the multi-bank controller to connect the sense amplifier to the PE. As above, the the numbers 0 to 3 correspond to the numbering of the sense amplifier. Bank-enable is a signal used for connecting control signals to the sense amplifiers, and row decoding, such that 'H' indicates PE control and 'L', indicates external memory R/W control. The bank-clock enable signal indicates that the internal address should be incremented because the program cycle has completed and is proceeding to the next cycle.

Table 4.2: Multi-Bank Controller Operation

Time	Bank-clock Enable	M Bank Address	Internal Address	M Register				Bank-enable			
				0	1	2	3	0	1	2	3
0	0	0	0	L	L	L	H	L	H	H	H
1	0	1	0	L	L	H	L	L	H	H	H
2	0	2	0	L	H	L	L	L	H	H	H
3	0	3	0	L	L	L	L	L	H	H	H
4	1	3	1	L	L	L	L	↑	↓	H	H
5	0	0	1	H	L	L	L	H	L	H	H
6	0	2	1	L	L	H	L	H	L	H	H
7	1	2	2	L	L	↓	↑	H	↑	↓	H
8	0	1	2	H	L	L	L	H	H	L	H
9	1	1	3	↓	↑	L	L	H	H	↑	↓
10	0	1	3	L	H	L	L	H	H	H	L
11	1	1	0	L	↓	↑	L	↓	H	H	↑
12	0	0	0	L	L	L	H	L	H	H	H

Note that when the M bank address is maintained during the activation of the bank-clock enable, that the sense amplifier being connected to the PE transitions from left to right as in Table 4.1. It should be noted that all control signals should be disabled before the transition is made; otherwise, unwanted operations on the memory could be performed. The bank-enable signals also transition, such that the sense amplifier with the same index as the internal address transitions from 'L' (low) to 'H' (high), and the sense amplifier with the same index as the new internal address transitions from 'H' to 'L'.

From Table 4.1 and Table 4.2, the multi-bank memory controller that can generate the bank-enable signalling is created, which is used for connecting the appropriate control signals to the proper banks of memory. In Table 4.2, the signals for the M register and bank enabling are shown. This table indicates how the M bank address, as well as the internal addressing, affects the M register and bank enable signals.

In Figure 4.12, the schematic for generating the bank-enable and M register selection signalling is shown. The figure was designed using the data from Table 4.1 and Table 4.2. This is one method; other methods are possible that use multiple counters. From the table, it is possible to see that the bank-enable signal is low where the internal address is equal to the number and is high for every other point, which is equivalent to the NAND operation. Similarly, when the M bank address, which is supplied externally to select which sense amplifier is to be connected to the PE is '3', all M Register select signals are low, indicating an invalid address. When the M bank address matches one of the values in Table 4.1, the M register select line will activate. From the internal address, it is possible to see that a method of addition would be needed. From M Register and Bank-enable, more than one counter or method of storing more than one value of the count is needed, since there are four signals, each which rely on the internal address. It was discovered that a cyclic register would be the best method of emulating the internal address, since there is no need to wait for the signals to propagate through multiple counters. Additionally, using multiple counters requires a register to hold the internal address and latch the new internal address when the bank-clock-enable signal is activated. The internal address can be broken into two cyclic registers, so that only a single shift is required instead of shifting by two.

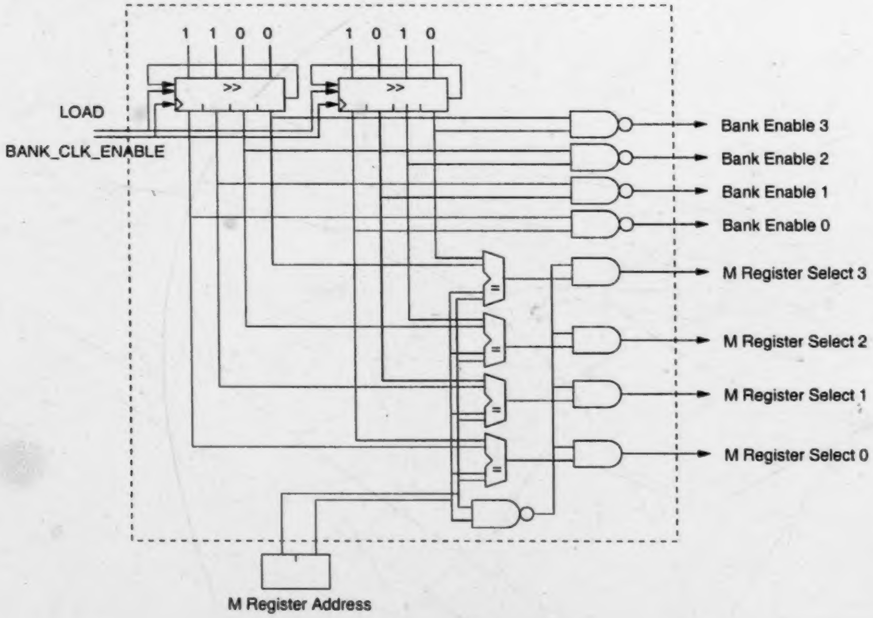

1 1 0 0
1 0 1 0
>>
>>
LOAD
BANK_CLK_ENABLE
Bank Enable 3
Bank Enable 2
Bank Enable 1
Bank Enable 0
M Register Select 3
M Register Select 2
M Register Select 1
M Register Select 0
M Register Address

Figure 4.12: Multi-bank Memory Controller

The cyclic shift register acts as the internal address; however, instead of incrementing an address, the addresses are pushed through the shift register. When both shift register bits are '1', the position of the bits is equal to an incrementing address. By using a cyclic shift register, the M register addressing is also implemented, thereby reducing the design complexity and increasing the speed of the design. It is also advisable that the shift register be able to load a default on power up, or when the user requests that the internal addressing be reset.

4.2.3 Memory Redundancy

The redundancy for the memory could be improved by adding redundant memory rows to the array, since column redundancy is already added through the PE redundancy. This type of column redundancy, however, is limited, because if a column of memory is found to have a fault and the PE associated with it is good, the good PE and the faulty column of memory must be bypassed. This by itself is not entirely bad, but if another PE in the redundancy block is found to have a fault and its memory column is found to be good, the block of redundancy will be identified as a failing block. For this reason, the introduction of the column redundancy technique, seen in Figure 4.13, was proposed so as to allow for the failure of one PE and one column in the redundancy block. A failure in the memory and a failure in the PE can be repaired using the proposed redundancy mechanism.

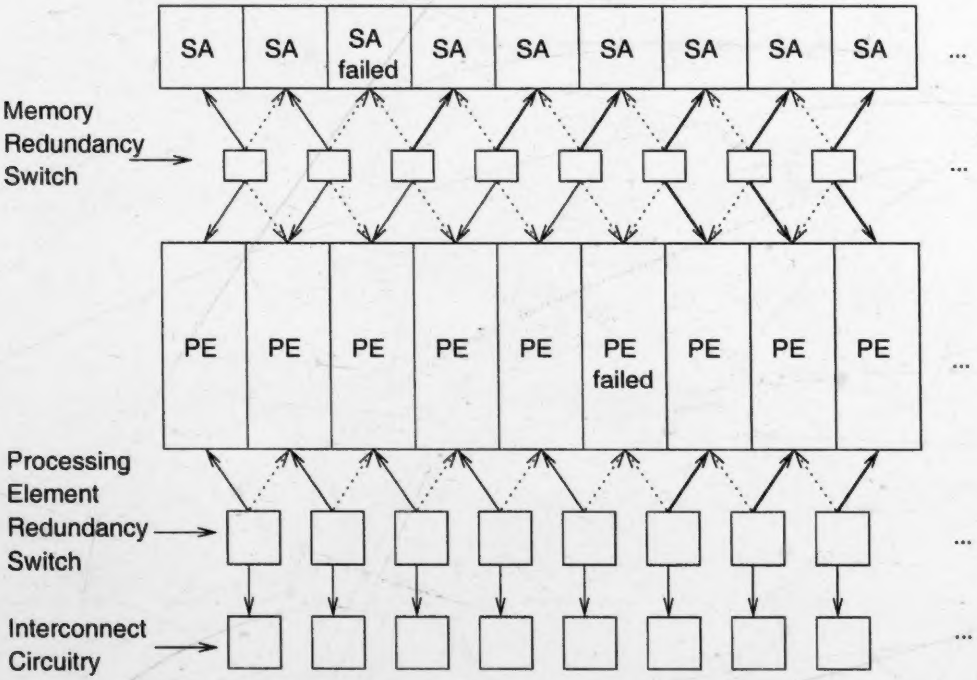

SA
SA
SA failed
SA
SA
SA
SA
SA
SA
...
Memory Redundancy Switch
...
PE
PE
PE
PE
PE
PE failed
PE
PE
PE
...
Processing Element Redundancy Switch
...
Interconnect Circuitry
...

Figure 4.13: Memory and PE Replacement

In Figure 4.14, the detailed schematic of the memory redundancy is seen. In the figure, the SA (sense amplifier) can be a single sense amplifier or a group of sense amplifiers, as in the multi-bank design. In the schematic, a ground is sent into the first switches of the redundancy group so that the first switch is not automatically pointing to SA1. In this way, the first SA or PE that fails will cause the remaining switches to change, thus pushing the the remaining switches over, as seen in Figure 4.13. Each sense amplifier or sense amplifier group contains an additional register used to store the skip result.

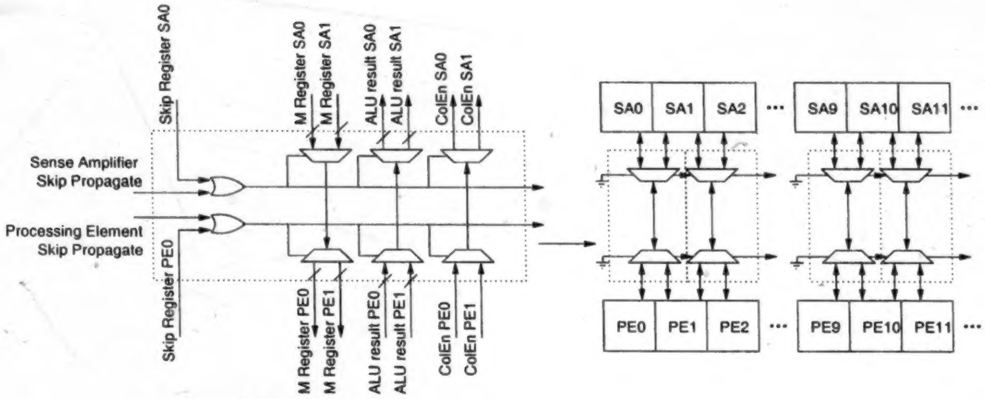

Skip Register SA0
M Register SA0
M Register SA1
ALU result SA0
ALU result SA1
ColEn SA0
ColEn SA1
Sense Amplifier
Skip Propagate
Processing Element
Skip Propagate
Skip Register PE0
M Register PE0
M Register PE1
ALU result PE0
ALU result PE1
ColEn PE0
ColEn PE1
SA0
SA1
SA2
...
SA9
SA10
SA11
...
PE0
PE1
PE2
...
PE9
PE10
PE11
...

Figure 4.14: Memory Redundancy Schematic Detail

4.3 Processing Element

The processing element in C•RAM is a 256-function ALU with input from three data registers. The 256-function ALU is created using a 8-to-1 multiplexer controlled by the three data registers X, Y and M. The M register is the sense-amplifier, with its contents being the currently accessed row of the memory. The X and Y registers store temporary values during calculations. The result of the ALU can then be stored to memory (M), to the X or Y register, broadcasted to all other PEs, or shifted to a nearest neighbour. The broadcast bus used in C•RAM is a wired-AND bus; all PEs must be broadcasting a '1' to produce a '1' on the broadcast bus. The broadcast bus is also read by each PE for use in other calculations. Each PE also contains a WE (write-enable) register which is used for performing parallel conditional operations, such as 'if' statements. It is possible to perform conditional nesting by writing the previous results of the conditional statement to memory, which are then backed out of as the nesting concludes.

The C•RAM processing element for this design is similar to the 3DSOI C•RAM PE, which has the additional skip register used to perform redundancy. The skip register can remain in the PE, or it can be moved into the redundancy mechanism; however, it was moved to the redundancy mechanism, since there is no reason for it to remain in the PE. It was originally placed in the PE because the redundancy

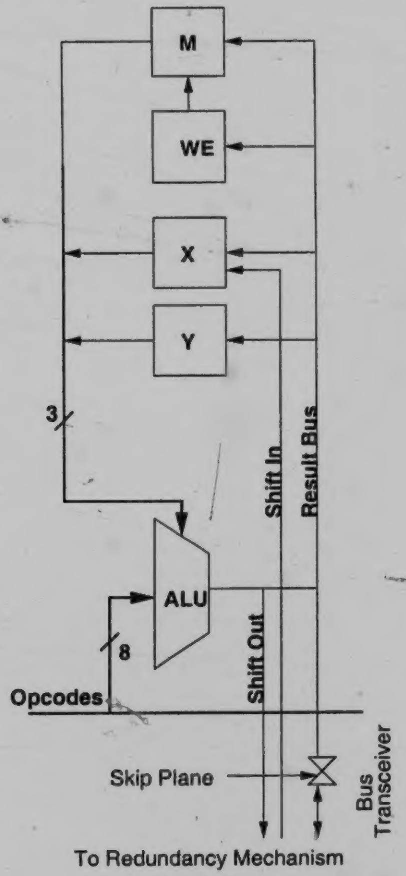

M
WE
X
Y
3
Shift In
Result Bus
ALU
8
Shift Out
Opcodes
Bus Transceiver
Skip Plane
To Redundancy Mechanism

Figure 4.15: Processing Element

mechanism was part of the PE. It was decided that the redundancy mechanism would be designed to perform replacement automatically, such that the setting of the skip register would automatically propagate a redundancy setup through the redundancy matrix. This reduces the amount of setup that is needed to perform redundancy, since the skip register can still be set by the results of the PE. This will be discussed more in the next section.

The PE was changed to allow a single register to be the destination of the shifted data, so the opcode would not need to be changed to alter the direction of shifting. It should be noted that this change can introduce non-symmetric timings; therefore, during design and simulation, the destination register was used to determine worst case timings. Since the shifting is no longer limited to only left and right, the shifting mechanism was taken out of the actual PE and was designed as a separate interconnection network. The shifting of data is then no longer dependent on PE failures, but on the way in which PE failures can be repaired. In this design, C•RAM is a three-dimensional mesh of PEs, and therefore may need to have redundancy for all the dimensions that the PE interconnect extends. The final choice as to how much redundancy is included will be determined by yield versus cost analysis. All dimensions of redundancy are designed in the following section.

4.4 Processing Element Redundancy

The basic PE redundancy allows the network to determine if each PE is an active participant in the computations, which requires that each PE be associated with a register indicating its activity. This register is called the S (skip) register for simplicity. The additional register is used since the WE register is used only for writing back to memory, and is used during conditional operations. The S register, however, keeps its contents until changed or until power-down. The S register is also used to control switches used for performing redundancy; in this way, the PE will either be connected to the flow of data or not. For example, if the S register for the PE is set, the broadcast bus signals will not be connected to the global broadcast bus. Therefore, the PE will not be able to be connected to the global flow of data through the broadcast bus. In Figure 4.16, the mechanism for PE redundancy is shown where five identical PEs are used to emulate four PEs. Thus, if one of the five PEs fails, there are still four functional PEs with the ability to perform proper data shifting. Since the shift data is sent through the redundancy switches, there will always be four PEs with valid shift data that will be able to emulate four sequentially ordered PEs.

4.4.1 Row & Column Redundancy

In Figure 4.17, a virtual 3x3 PE grid exists. In this, each plane is able to repair itself, then each virtual PE can be connected to the virtual PEs on another plane; in this way, the three-dimensional interconnection network remains intact. Note that the

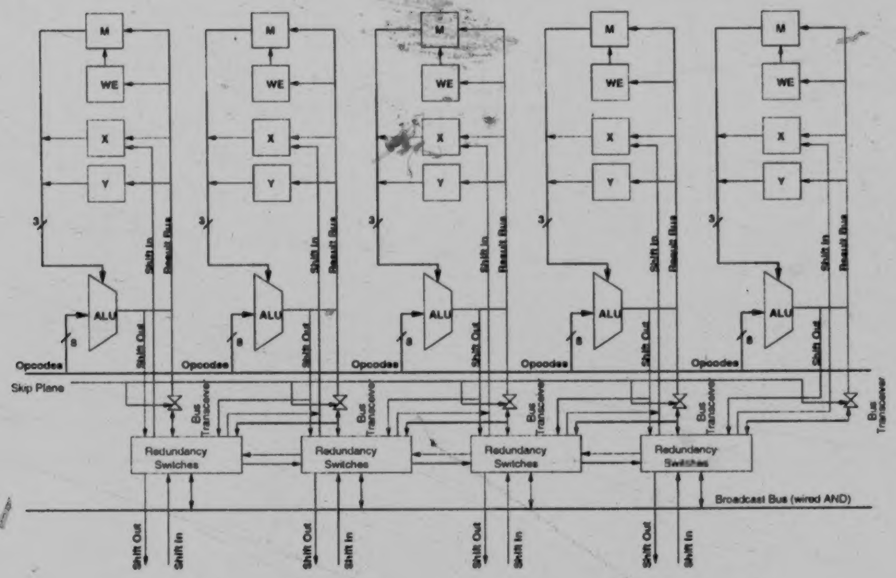

M
WE
X
Y
ALU
3
8
Shift In
Result Bus
Shift Out
Opcodes
Skip Plane
Bus Transceiver
Redundancy Switches
Broadcast Bus (wired AND)
Shift Out
Shift In

Figure 4.16: Processing Element Redundancy

redundancy method does not require that all planes be repaired in the same way. It should also be noted that if a plane contains too many failures such that the failures can not be repaired, there exists another level of redundancy that is able to bypass the planes that fail.

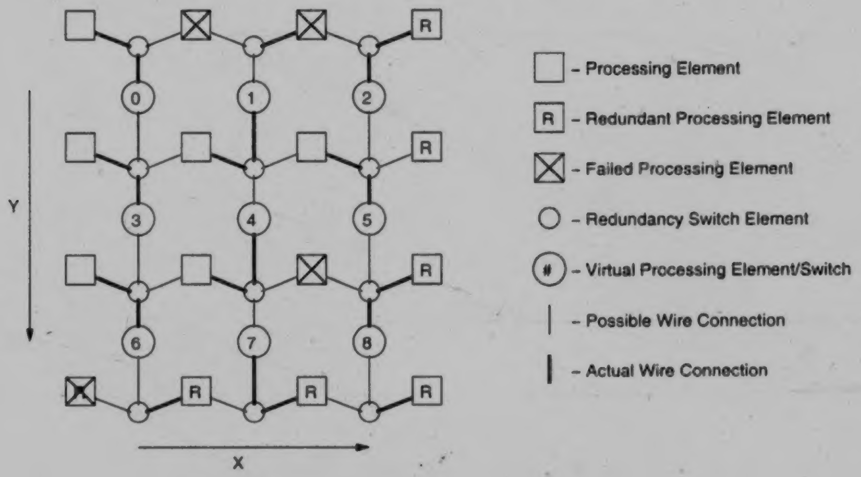

R
0
1
2
3
4
5
6
7
8
Y
X
- Processing Element
R - Redundant Processing Element
- Failed Processing Element
- Redundancy Switch Element
# - Virtual Processing Element/Switch
- Possible Wire Connection
- Actual Wire Connection

Figure 4.17: Two-dimensional PE Redundancy

In Figure 4.18, the schematic representation of the row and column redundancy method is shown. Here, the circuit translates a two-dimensional grid into a linear array attached to a memory array, since a memory array is organized linearly. The figure also shows how the actual processing elements are turned into virtual processing elements through a processing element addressing scheme. It can also be seen that the interconnection network is connected to virtual processing elements rather than the actual processing elements. This removes the need for additional circuitry to perform redundancy in the interconnection network. All the redundancy is applied to the processing elements and memory, since they are the most likely place for defects to occur.

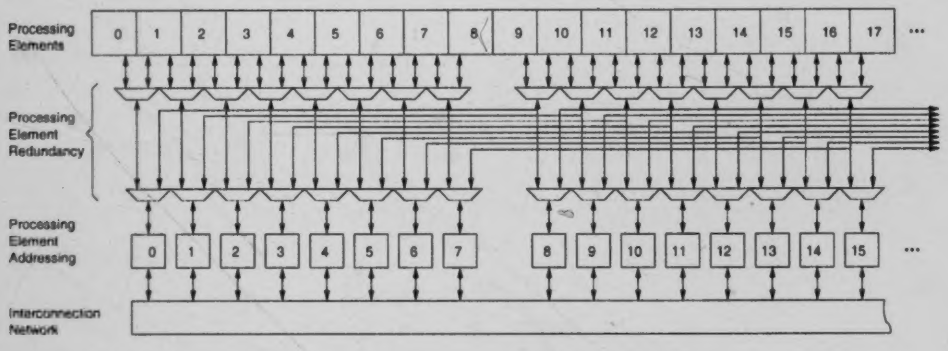


Figure 4.18: PE Grid Redundancy Schematic

A detailed schematic of the redundancy is shown in Figure 4.19, where the flow of data is through a selection matrix controlled by the S register, not shown. The results of the S registers are combined to create the additional signalling required to automate the replacement of failed PEs. This combination of S registers can also be used to create the signal for the failure of a plane of PEs.

4.4.2 Redundancy Controller

The redundancy controller is the circuitry used to determine the failure of a PE or the failure of a plane of PEs. The plane failure is determined by taking the skip signals of the columns of PEs and combining them to determine if the plane has failed. The recomputing of the address of each plane in the design is performed using the results of the plane failure identification, as seen in Section 4.2.1.3. The

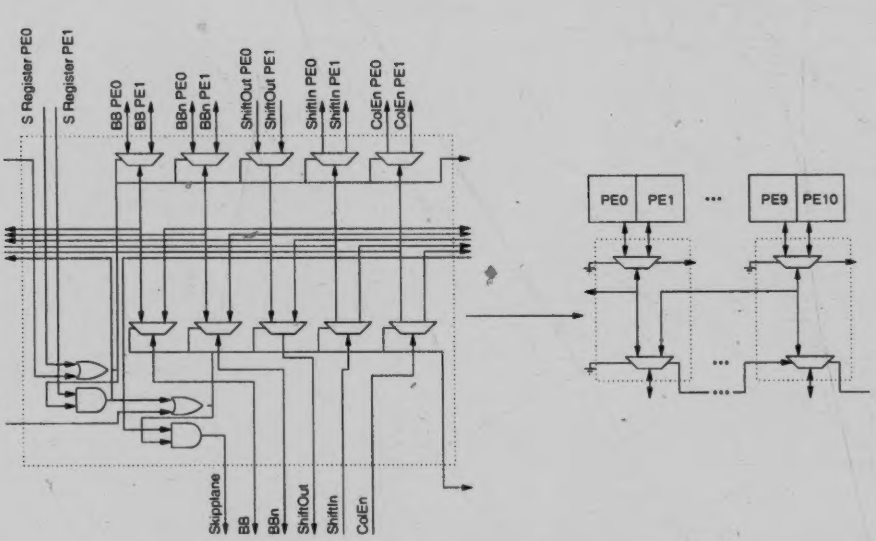

S Register PE0
S Register PE1
BB PE0
BB PE1
BBn PE0
BBn PE1
ShiftOut PE0
ShiftOut PE1
ShiftIn PE0
ShiftIn PE1
ColEn PE0
ColEn PE1
PE0
PE1
...
PE9
PE10
Skipplane
BB
BBn
ShiftOut
ShiftIn
ColEn

Figure 4.19: PE Grid Redundancy Detail

recalculation of plane addresses is done in order to maintain uniform addressing for external accesses. By maintaining uniform external addressing, the programmer or user does not need to be informed of which PEs or planes have failed. The redundancy controller performs repair (replacement) automatically, thereby removing the responsibility from the user.

4.5 Interconnection Network

All C•RAM architectures have implemented some type of an inter-processor communication network (interconnection). There have typically been two interconnection networks in C•RAM: the one-dimensional patterned left-right communication, and the broadcast bus. The left-right communication is used for shifting data around the C•RAM array, while the broadcast bus is used for sharing data between all the PEs simultaneously. For this research, the broadcast bus is changed by the effects of redundancy, while the one-dimensional patterned communication is extended to allow for two-dimensional and three-dimensional patterned communication between processors. Because the MIT LL process is used, the interconnection network can be divided and/or built to be scalable for another dimension. In this way, the interconnection network must be able to work when only one die is in the stack, as well as being able to accommodate stacking of extra dies. The stack must maintain one-dimensional, two-dimensional and three-dimensional interconnections.

4.5.1 Broadcast Bus Interconnection

The broadcast bus must also be workable for multiple dimensions of interconnect; most specifically, for multiple dies being stacked. The broadcast bus must be able to tie multiple dies' broadcast buses together to allow broadcast bus communication to occur between the dies in a stack. The broadcast bus circuit developed in 3DSOI C•RAM already accommodates broadcasting between multiple chips, so there is no need for changes to be made to allow for broadcasting between dies, since it is already available. It is also possible to create a segmented broadcast bus where each plane of PEs can broadcast between only the PEs of its own plane; however, it was decided that this modification would not be incorporated. The segmented broadcast bus was not incorporated since there did not seem to be any forseeable advantage in doing so, and the design issues that it introduced also contributed. Some design issues were connecting the plane segmented broadcast buses externally for reading and writing, so that as the number of planes grows, so does the number of pins required for all the broadcast buses.

Due to the redundancy design considerations, a small change to the broadcast bus was performed. In order to ensure that only PEs with valid data are connecting themselves to the broadcast bus, it was required that the connection to the broadcast bus flow through the redundancy circuitry. In this way, the broadcast bus performs its task properly and the invalid PEs do not interfere with correct operation of the broadcast bus. The broadcast bus flows through the redundancy circuitry in case all the PEs are good, causing the redundant PEs to broadcast data onto the broadcast bus when they do not have valid data. When the plane that a PE is located on fails, all the PEs on that plane must have their broadcasting privileges severed. To perform this, the signal used to indicate the failure of a plane is passed back to all its PEs; if the signal is low, all the PEs can broadcast their data; otherwise, they will all be disconnected from the broadcast bus. This is to ensure that failed PEs will not interfere with proper operation of the broadcast bus.

As 3DSOI C•RAM already incorporated the disabling of the broadcast bus transceiver, this does not need to be changed. A change, however, is made to the signal that disables the transceiver. In 3DSOI C•RAM, the transceiver is disabled by the S (skip) register. In this design, the S register disconnects PEs locally from the broadcast bus, and the plane fault signal disconnects PEs globally from the broadcast bus.

4.5.2 1D, 2D, and 3D Interconnection

The previous chapter showed that a one-dimensional and two-dimensional interconnection network could be embedded into a three-dimensional interconnection network. It was also shown that to implement PE redundancy and maintain the interconnection networks, the interconnection needs to be separated from the PE and its redundancy. In this section, the circuits used to implement the interconnection are designed. The shift circuit is implemented in such a way that each PE communicates with the shift circuit and each shift circuit can communicate with six other possible shift circuits. All the shift circuits are controlled by the same signals so that they all shift left, right, north, south, up or down. This is where the patterned communication enters into the design. The shift circuit, however, is only part of the solution. The shift circuit is easily used to create the three-dimensional mesh of PEs, but it is necessary to incorporate the one-dimensional and two-dimensional networks as well. This required additional circuits on the boundaries of the three-dimensional mesh that are used to transform a three-dimensional mesh into a one-dimensional string of PEs or a two-dimensional grid of PEs. These circuits also make allowances for failed planes of PEs, and then bypass the failed planes by connecting only valid planes together.

It should be noted that within each of the dimensional modes it is possible to use the properties of the mesh to perform long shifts. For example, if the mesh is an 8x8x8 PE array and the program is operating in one-dimensional mode, a left shift of 64 PEs is required. This could be done by shifting down one plane of PEs instead of shifting left 64 times. The design of the interconnection network and the interconnect controller takes this example into consideration, due to the dynamically reconfigurable nature of the interconnect.

4.5.2.1 Shift Circuit

The shift circuit is designed to allow PE redundancy to be performed without affecting the inter-PE connection. The PE redundancy is built so that the data shifted in and out from the PE is sent through the redundancy matrix before being used by the shift circuit. In this way, the shift circuit only needs to be concerned with connecting the data being shifted out to the proper interconnect wire, as well as connecting to the proper interconnect wire to allow for data to be shifted in. In Figure 4.20, the circuit to perform this task is shown. The shift signals are buffered to ensure that the signal has enough drive, since the switches will experience a threshold voltage (V_T) drop for each switch that the signal passes through. Since there will be a drop of approximately two times V_T, the inverter for the shift in data may consume static power when a '1' is being shifted. This is attributed to the problem NMOS transistors have in passing a '1' (V_{DD}). Since, the signal being buffered into the PE is less than the full V_{DD} the inverter will consume power due to the crowbar effect. The crowbar effect is seen when there is a direct path from V_{DD} to ground. This occurs because the PMOS transistor of the first stage of the buffer will not be completely turned off. The shift circuit is composed of a number of switches which are connected to the interconnect wires.

In Figure 4.21, a selection circuit (MUX) is added for the shift out data. This is to accommodate the possibility of a failed plane when shifting in the z direction (up/down). The MUX will redirect the shift in data to the output if the plane is to be skipped, and the shift operation being performed will be in the z direction. The redundancy for the z direction was added because of its proximity to the shift circuit, thereby reducing the delay penalty that a failed plane injects.

It should be noted that only three actual interconnect wires are required to per-

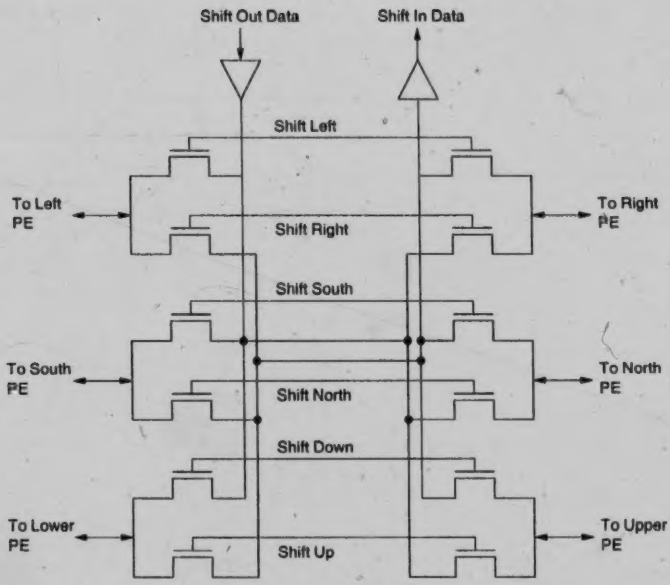

Shift Out Data
Shift In Data
Shift Left
To Left PE
Shift Right
To Right PE
Shift South
To South PE
Shift North
To North PE
Shift Down
To Lower PE
Shift Up
To Upper PE

Figure 4.20: Shift Circuit

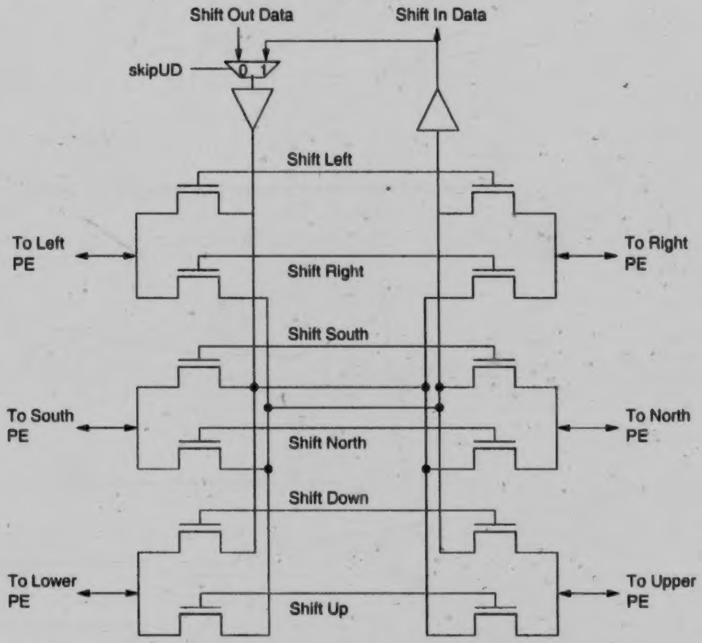

Shift Out Data
Shift In Data
skipUD
0
1
Shift Left
To Left PE
Shift Right
To Right PE
Shift South
To South PE
Shift North
To North PE
Shift Down
To Lower PE
Shift Up
To Upper PE

Figure 4.21: Shift Circuit with z Redundancy

form six dimensions of communication. In Figure 4.20 and Figure 4.21, six connection wires are shown; however, these six connection wires can be combined into only three interconnect wires. In Figure 4.22, the interconnection of the shift circuits is seen for a 2D grid. In the figure, we see that the Ls (to Left PE) are bi-directionally connected to the Rs (to Right PE), and the Ns (to North PE) are bi-directionally connected to the Ss (to South PEs). This can be extended to the 3D mesh in the same way. The implementation uses the bi-directional capability of a wire to perform bi-directional communication, thereby reducing the number of wires that need to be routed. Due to the size of the PE mesh, the number of wires used for interconnect could be quite high if each direction of communication had its own wire. In this way, the number of wires is reduced by a power of 2. It should also be noted that by using the MIT LL technology, the number of wires by are reduced by yet another factor. When the $N \times N \times N$ sized mesh is divided into X dies, the number of wires on the die are reduced by a factor of X. This is possible because the MIT LL technology has additional vias that can connect metal layers between dies. The number of wires routed horizontally or vertically on a die are reduced since the wire is now created through the dies, reducing wire length and the delay path for a signal.

4.5.2.2 Fault-Tolerant Network Interconnections

To implement the one-dimensional and two-dimensional interconnection within the three-dimensional mesh, additional circuitry is added on the boundaries of the mesh to form the interconnection networks. This would appear to be quite simple if not for the redundant nature of the design. The boundary circuits were designed to respond based on whether the plane of PEs is still good or not; therefore, the circuit is able to repair the connections between planes so that the bad planes of memory can be bypassed to maintain the proper interconnections. It should also be noted that these boundary circuits are only active when the proper interconnection mode is entered.

Since all the dimensional modes can have common interconnection modes such as a spiral interconnect used for left-right interconnect in two-dimensional and

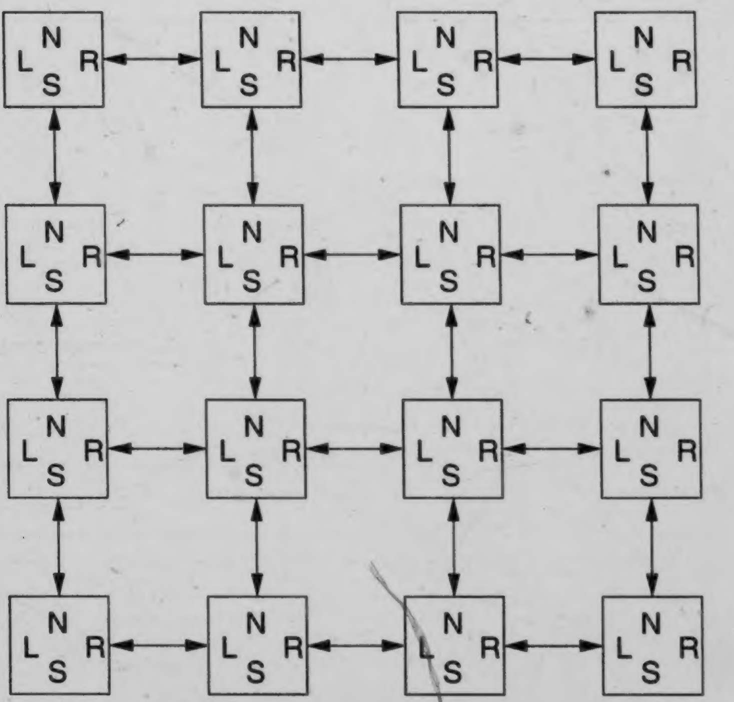


Figure 4.22: Shift Circuit Interconnection

three-dimensional, the wiring was designed to reduce the amount of additional wiring and circuitry used for the boundary interconnect. For one-dimensional and three-dimensional interconnect, the spiral interconnect was designed to be identical, and therefore, the boundary circuit for the spiral is common, as seen in Figure 4.23.

For two-dimensional interconnect, using a spiral interconnect for left-right communication presents two possibilities. The first possibility is to use the spiral interconnection of the one-dimensional and three-dimensional, as seen in Figure 4.24(a). This results in only one spiral network for 1D, 2D and 3D, and maintains sequential PE addressing. This method also requires that an additional boundary interconnect circuit with redundancy be built for north-south interconnections. In the other possibility, the two-dimensional grid connect and one-dimensional spiral connect share many of the same interconnections. The only changes would be on the boundary of the two-dimensional grid, where the interconnects would need to be staggered for

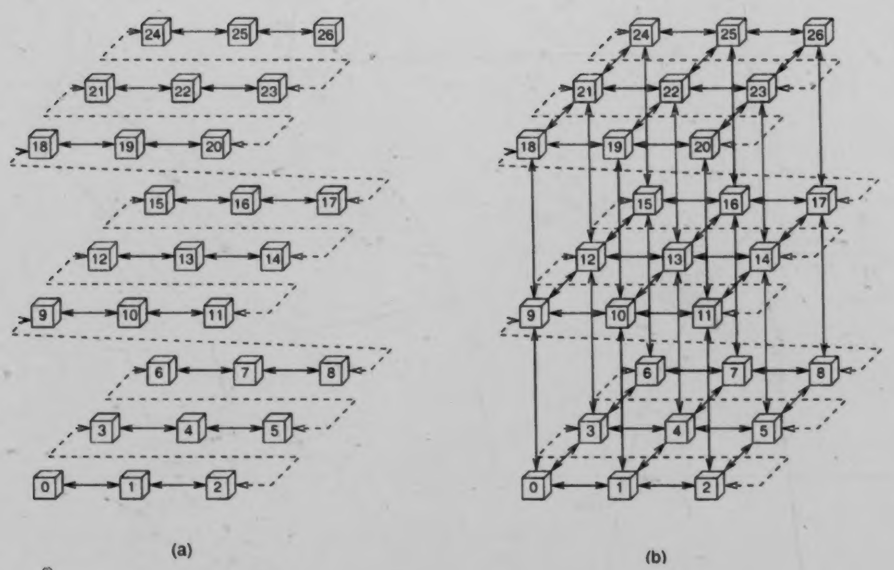


Figure 4.23: Interconnections for 1D and 3D using Spiral

the spiral interconnect; see Figure 4.24(b). The PE ordering for this second method is no longer sequential and requires more complex programming and data writing techniques. This also interferes with the ability of switching between the 1D, 2D and 3D interconnection schemes, and maintaining PE interprocessor relationships.

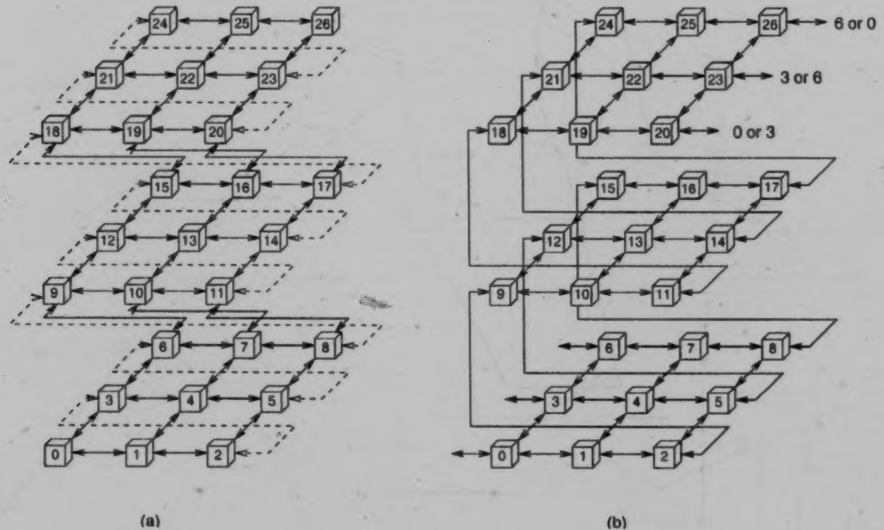
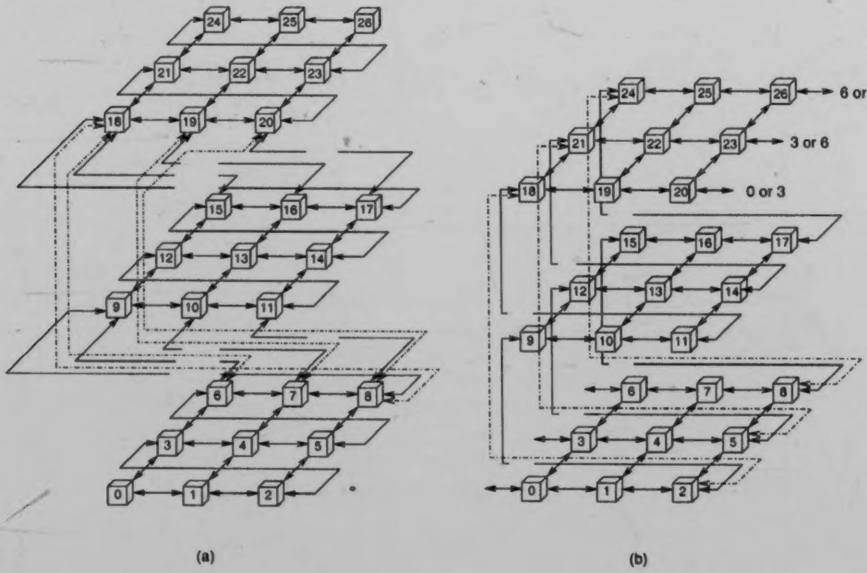


Figure 4.24: Interconnections for 2D using Spiral

To maintain the interconnection during a failure of a plane, the failed plane must be bypassed with a minimal amount of effort and circuitry. In Figure 4.25, the plane interconnects must be severed and replaced by a new interconnection. This results in additional circuitry that may add extra delay to the interconnection chain; however, this is necessary to maintain full functionality of the interconnect in the event of a plane failure. Looking at Figure 4.25, it would appear that Figure 4.25(b) is less complicated that Figure 4.25(a); yet the amount of additional circuitry is the same for both. For this reason, the two-dimensional spiral option that was chosen was Figure 4.25(a), so that the PE ordering would be maintained for a left-right spiral interconnect.



(a)

(b)

Figure 4.25: PE Interconnections with a Failed Plane

The circuit in Figure 4.26 is the method designed to create the interconnections between planes for a spiral interconnect. This circuit has the benefit of implementing the redundancy and the standard interconnect together. In the figure, the blocks are shift interconnect circuits that perform the actual shifting of data into and out of the PE. The **vPlane(0,1,2)** signals indicate a valid (good) plane, and if the spiral is to be closed so that PE interconnect 26 is connected to PE interconnect 0 then **ClosedSpiral** is asserted. If **vPlane1** is low, indicating a failure, then the data of plane one is bypassed, and PE interconnect 8 is connected to PE interconnect 18 through the elements: Q1, I0, Q5, I1, and Q9, where I0 and I1 are bi-directional tri-state drivers controlled by the direction of the shift. Using the tri-state drivers, the shifting delay can be reduced, since stand-alone NMOS and PMOS transistors will reduce speed by reducing the drive of a signal. In this circuit, if all the planes are valid (no failure), then the data is appropriately shifted to the proper interconnect block of the next plane. If a plane is found to have a failure, the interconnect blocks from the planes above and below the failed plane are connected together instead. In this way, the failed plane data is bypassed, thereby removing the possibility of the data interfering in the computations.

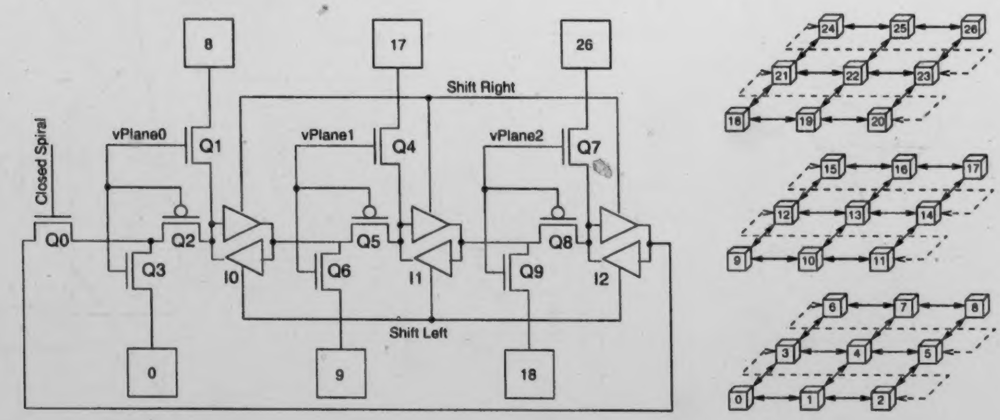


Figure 4.26: Spiral Interconnection Circuit with Redundancy

4.5.2.3 Torus Interconnection

Since most early 2D interconnected PE arrays have included the torus interconnection construct, it was decided that the torus interconnect would also be included. The addition of the torus to the network is done to finish the interconnection network. The torus is used to close all dimensions of communication so that all the data will remain in the mesh. When looking at the cube structures throughout this chapter, it can be seen that the spiral interconnection and torus interconnections will require extended routing for PEs that are located on the boundaries of the cube. This would result in a longer delay for the signal to reach its destination, meaning the entire shift would have to be slowed down to the longest delay. In order to produce nearly identical communication delays when communicating through the torus, the organization in Figure 4.27 is used to route the interprocessor communication wiring. This method of folding the torus was performed in the Cray T3D supercomputer. This method increases the delays between individual processors, but it also reduces the delay for communicating between the boundaries. This means that the shift circuit can actually operate at a faster frequency despite increasing the delay between the PEs in middle of the torus. This also reduces the delays that would be present in the boundary circuits since the boundaries of the mesh that would need to be closed to complete a torus would all be near each other. All the left-right boundary PEs would be located side by side, as would the north-south and up-down PEs. This also produces the effect that an entire array could be replicated and the arrays could be tied together, and the torus could still be complete; see Figure 4.28.

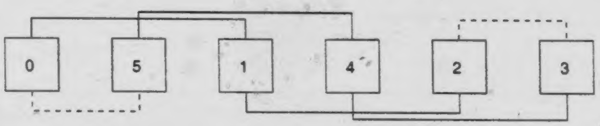


Figure 4.27: Torus with Similar Delay Links

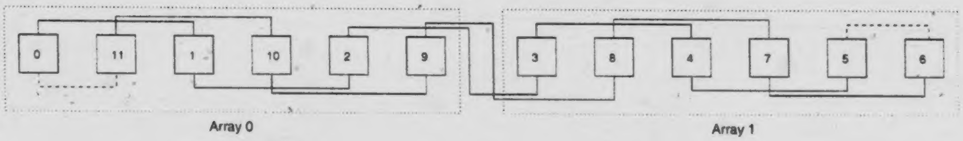


Figure 4.28: Torus Across Multiple Arrays

In the above figures, the right end of the torus can be extended, but at some point it will need to be closed in order to maintain the interconnection. This closing of the right end of the torus could be controlled by a switch that is enabled when the array block is found to be at the right-most boundary. Similarly, for the left interconnection, the closing of the torus should only be possible when the array block is at the left-most boundary and the method of interconnection is a torus. It should be noted that this mechanism is implemented for all dimensions of the PE mesh, thus reducing the delay for PEs located on the boundaries of a mesh.

In Figure 4.28, the PE ID numbers progress across all the arrays. From this it was determined that a measure of address translation was needed if more than one array block was used. To accommodate multiple array blocks, an address translation circuit was designed that translates the linear logical address into a physical address corresponding to the method of PE ordering shown in the figure. Since the number of rows and columns of PEs per plane in an array block are known and those numbers are divided in half for the torus, a parameterized verilog model was constructed that can be modified to be synthesized for any number of blocks on the die. For example, in the figure, if the logical address input into the chip was 10, the circuit translates this into an enable signal for array 0, and the physical address 4 of the array.

4.5.2.4 Array Block Interconnection

As discussed in the previous section, the design is created to be scalable through the creation of a scalable torus architecture that is able to connect two blocks together to form a new, larger array of PEs. This scalable torus architecture is not confined to only a single die, but is extended to multiple dies through the use of MIT LL's 3D SOI technology. For this reason, each block will be designed identically. To ensure that each block will not interfere with the total torus, boundary identification must be added in order to produce a cohesive 3D mesh. This means that only blocks that meet the boundary conditions should close the points on the boundary. For example, only blocks that are identified as being the right-most block should be allowed to close the right side of the torus. In Figure 4.29, the boundary connections for left-right interconnection network is shown where the left-right interconnection is built from three identical array blocks. The L and R signals should be hardwired appropriately to create the proper interconnect. In the figure, the dotted-box represents an array block, and the torus control signal is external to the array block, while the block boundary position identifiers are local to the array block.

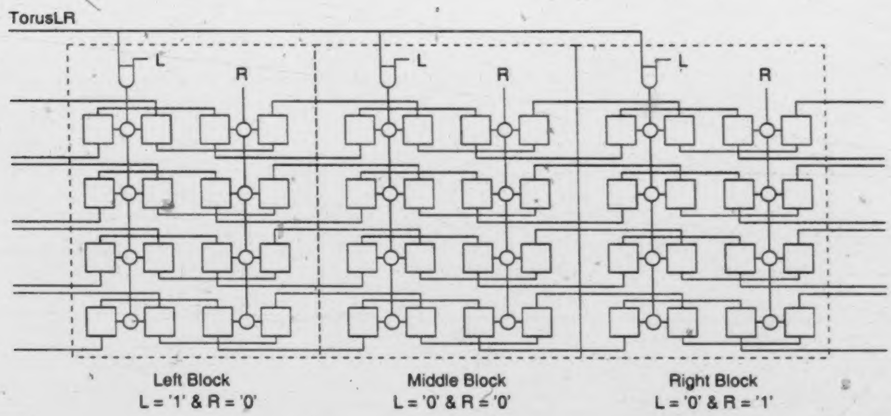


Figure 4.29: Array Block Interconnection over Multiple Array Blocks

This mechanism is extended to the north-south and up-down interconnection network; however, the up-down (3D) interconnection uses a special boundary identifier due to the stacking ability of MIT LL's 3D SOI technology. This technology requires that the stack be able to automatically identify the top and bottom die of the stack, since it is possible to have a single die or a stack of two or more dies. For left-right and north-south interconnect, the single die must be complete in that all the array blocks must be placed and that each array block is assigned the proper boundary identifiers corresponding to its' place in the left-right and north-south interconnection grid.

4.5.2.5 Boundary Interconnections

Previous sections have discussed the fault-tolerant nature of the interconnection network using additional planes of PEs. This section discusses how multiple interconnection networks can be formed within a 3D mesh of processing elements (PEs) by using the torus interconnection layout for producing similar delays for shifting between PEs. It was discussed previously that besides the 1D, 2D and 3D, additional elements could be added to the simple mesh to produce a 3D toroidal mesh. Similarly, a spiral network could be added into the 3D mesh to produce the 1D PE array, and this spiral network can be used in the 2D and 3D interconnection networks. Most of the discussion has been on how to perform the redundancy for the spiral network to maintain the spiral through the mesh in the event of a failure, but this is only one part of the spiral. The boundary interconnections must provide reconfigurability to switch between a left-right spiral interconnect, a left-right torus interconnect, or no left-right interconnect on the boundaries of the PE array. Multiple switches exist for the PEs on the boundary to connect PEs in the same row (torus) or PEs in different rows (spiral), or not to be connect to any other PEs. The boundary interconnections exist in all array blocks, and therefore, only the switches that exist on real boundaries perform boundary operations.

4.5.2.6 Off-chip Interconnection

To maintain the ability of scaling the processing power through the interconnection of multiple C•RAM chips, the shifting of data for all the inter-processor communication networks is supported. This is implemented by creating three groups of pins: one group for left-right communication, one group for north-south communication, and one group for up-down communication. These pins should be bi-directional, since internally, all interconnections are bi-directional to limit the number of pins required for shifting. The design of this research is based on a chip package containing an 8x8x8 cube. Since the design uses the MIT LL 3D SOI technology, each die contains an 8x8x4 cube. From this it was determined that the off-chip shifting would require eight pins to be used as the shift bus width. This means that 16 pins are used for left-right interconnection, 16 pins for north-south, and 16 pins for up-down. Therefore, the total number of pins being used for shift interconnect is 48.

The off-chip interconnection circuit is designed to support all possible inter-processor communication networks. For instance, when the chip is in 1D communication mode, only one PE should be receiving data from off-chip and only one PE should be sending data off-chip. It should be noted that all PEs can send data off-chip without interfering with operations, but if all PEs are receiving data from off-chip, it will interfere. Therefore, the receiving circuit is the only circuit that needed special consideration. It must, however, be noted that 1D communication is special since the (N,N,N) PE does not line up with the (0,0,0) PE in the shift bus.

From the discussion above, it can be seen that the off-chip interconnection is dependent on the communication network type. Thus, a controller was designed that sets up the off-chip interconnection based on the communication type and current state of shifting. It should also be noted that 1D off-chip shifting does not maintain the same PE ordering that is found in 2D and 3D off-chip shifting.

4.5.3 Interconnection Network Controller

The interconnection network controller controls the method of interconnection to be used for inter-processor communication. There are six directions of communication that would require 6 pins for operation. Using a decoder built as part of the interconnection controller, this is dropped to 3 pins using an 8-bit decoding value. There are also multiple types of interconnection that are possible, such as torus connection for the left-right (east-west), north-south or up-down directions of communication, spiral interconnection for left-right shifting operation, closed or open spiral interconnection, and no boundary interconnection. Some, however, are mutually exclusive, and this is converted into a decoder circuit. The left-right spiral interconnect and left-right torus interconnect are mutually exclusive interconnection methods, so a decoder circuit was created that ensures that these two interconnection methods are not enabled simultaneously. If both are enabled, there will be contention between the signals being shifted into the boundary PEs.

There should only be one interconnection network controller for the entire die, rather than a controller for each array block. This is possible since all array blocks are shifting data in the same direction.

4.5.4 Off-chip Interconnection Controller

The off-chip interconnection controller controls the off-chip shifting circuits that are used for extending the inter-processor communication network over multiple chips. The controller only requires two pins in addition to the pins needed for shifting data in and out of the chip. The two pins for the controller are used to reset the controller and to increment the internal counter used by the controller to multiplex the shift data in and out of the chip. On the rising edge of the clock, the data is shifted out, and on the falling edge of the clock, the data being shifted in is latched. In this way, the multi-chip communication is synchronized so that the data of the chip is sent out, and given sufficient time to be set up before it is sampled by the chip looking at the shift data.

During the design of the off-chip interconnection controller, it was determined that the circuit designed for plane addressing could be used to maintain off-chip shifting in the event of plane failures. The controller could send a plane address to the plane address enable circuit, and if the plane address from the shift controller is identical to the address of the plane, the shift data for that plane would be selected for input and output.

As in the interconnection network controller, there should only be one off-chip interconnection controller for the entire die. There should only be one off-chip interconnect controller that is active for a complete package as well. This means that if multiple dies are stacked together, only one of the controllers should be able to set up the selection of the shift data.

4.6 3D SOI Stack Implications

When using MIT LL's 3D SOI technology, the ability to stack a number of identical dies brings up the problem of multiple controllers being active in the stack of dies. This can be repaired by the introduction of a master die select signal which can be connected to the outputs of each controller on a die. In this way, only the die that has the master signal active will control the operation of the stack. The completion of this scheme requires that the outputs of the controllers on the master die be sent down/up through the stack to ensure that all dies in the stack are operating under the same control. The introduction of stacking dies to create a scalable 3D C•RAM mesh requires that the 3D via (inter-die via) be properly connected in order to produce a scalable mesh while maintaining single die functionality. In Section 2.2.2, Figure 2.5 shows how a 3D via is formed between two dies; however, in the figure we see that the two dies are not identical. At the University of Alberta[16], a technique for stacking identical dies was created along with circuits for identifying the top and bottom die in a stack. A master die identifier was also developed, along with a symbolic representation for the 3D via. The 3D via symbol is representative of the layout of the 3D via used to stack identical dies. Figure 4.30(a), shows the symbol created at the University of Alberta to represent a 3D via that can be used in a stackable design. In Figure 4.30(b), the symbols (3D vias) are stacked to transfer signal data to/from a lower or upper die.

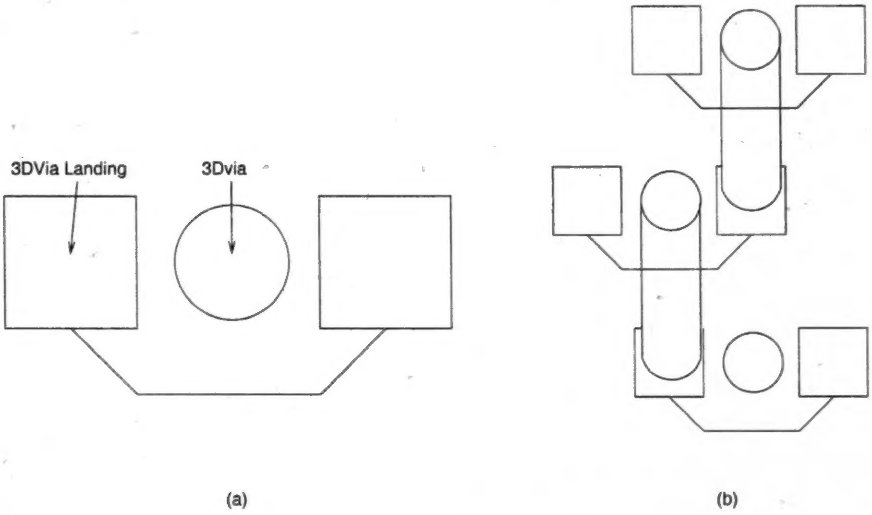


Figure 4.30: 3D via (a)symbol, and (b) stacking

It should be noted that the 3D via, just like a wire, is bi-directional; therefore, it can be used by the middle die to send or receive data from the die below. It is also useful in creating the identification circuits and conceptualizing the flow of global data, as well as inter-die dependent data.

4.7 Performance

The performance of this design was evaluated using Cadence to create schematic circuits and Hspice to perform the simulation of these circuits. The performance of the design is an approximation since it was found that the simulation of a complete

4x8x4 (x,y,z) PE array was not viable. Therefore, approximations were made by computing the capacitance and resistance of the wiring to determine wiring delays that could be expected for driving signals across the memory and PE array. Since SOI is a 1.5V process, the designing of the PE array was optimized to obtain the highest performance when running at 1.5V. The design was also simulated at 1.2V to ensure that the design would be operable at 1.2V, and to determine the performance reduction when operating at the lower voltage.

4.7.1 Design Performance

The following performance parameters are based on a cube where x equals 4, y equals 8 and z equals 4. It could be possible to place all the PEs on the bottom, since each PE is attached to 4 columns of memory. To reduce the routing congestion for the redundancy and shift interconnect, it is assumed that the PEs are placed above and below the memory array. The actual dimensions for the cube then become x(5), y(10), and z(4) when redundancy is included. Placing PEs above and below relaxes the pitch used for matching the PEs to the memory columns.

The simulation was performed using the lumped-RC T-model where the resistance and capacitance were extracted by creating layout. During the layout, wire lengths for the shift and redundancy circuits were obtained, along with an equation to estimate wire lengths for other array sizes. The maximum wire lengths for the redundancy and shift circuits and for the 4x8x4 PE array were used during the simulation of the redundancy and shift circuits.

The maximum wire lengths for the redundancy circuit are:

$$L_{wire} = (y_{dim}/2 + 1) \times z_{dim} \times w_{PE}$$

or:

$$L_{wire} = (nRows_{mem} + nRows_{redmem}) \times h_{SRAM}$$

The maximum wire length above is for creating the lumped-RC T-model used for simulating the redundancy delay.

The maximum wire lengths for the shift circuit are:

$$L_{wire} = (y_{dim}/2 + 1) \times z_{dim} \times 2.5 \times w_{PE}$$

or:

$$L_{wire} = (nRows_{mem} + nRows_{redmem}) \times h_{SRAM}$$

The maximum wire length above is for creating the lumped-RC T-model used for simulating the shift circuit delay. The complete shift circuit actually includes the redundancy delays since, a shift must pass through the redundancy circuit before it gets to the shift interconnect circuit. The shift must then pass back through the redundancy circuit to get back to the PE. It should be noted that the 2.5 factor was used to accommodate the possibility of connecting two array blocks together.

From Table 4.3, the minimum timing for ALU operations, shift operations, and memory operations are shown, along with timings for select circuits. The minimum timings for the circuits assume that signals can be supplied at precisely the right time to achieve maximum speed. This works in theory, assuming that the signal can be supplied at exactly the right time using precision self-timed circuits.

In the table, the shift operation takes approximately three times as long as an ALU operation. It should be noted; however, that the shift operation includes an ALU operation as well. The complete shift operation involves the ALU operation of setting the ALU result bus, then shifting out the result bus through the redundancy, through the shift circuit, back through redundancy, and into the X register. Therefore, the shift operation requires only two additional ALU cycles to complete.

The maximum operational speeds are shown when operating at 1.5V, and are summarized in Table 4.4. The maximum speed of operation, based on the ALU cycle, is 256MHz, assuming that signals can be supplied precisely. The maximum operational speed for writing memory externally is 121MHz or 121Mb/s, while reading is 129MHz. The maximum operational speed for writing memory internally for the PEs is 175MHz; or 22.4Gb/s bandwidth for all 128 PEs. Reading memory internally for the PEs results in a maximum operational speed of 148MHz.

The maximum operational speeds are shown when operating at 1.2V, and are

Table 4.3: Circuit Propagation Delay for 1.2V & 1.5V Operation

Circuit	Condition	Delay @ $1.2V$	Delay @ $1.5V$
Signal Drive Delay	Single Driver	$1.68ns$	$1.38ns$
Signal Drive Delay	Drive 2 sides	$1.26ns$	$968ps$
Boundary Interconnect with Redundancy	2 Sequential Failed Planes	$620ps$	$535ps$
Redundancy Switching Circuit	2 Level Redundancy	$2.65ns$	$1.55ns$
Shift Interconnect Circuit	Through Redundancy	$9.26ns$	$5.2ns$
Overwriting Register X, Y, or WE	Write Enable to Overwrite	$1.19ns$	$1.05ns$
Maximum Delay for Computing	Ext. Opcode to ALU result	$4.89ns$	$3.29ns$
Maximum Delay for Shift	ALU result to X register	$12.3ns$	$7.35ns$
SRAM Write Cycle Time	External IO Cycle Time	$10.8ns$	$8.2ns$
SRAM read Cycle Time	External IO Cycle Time	$9.64ns$	$7.74ns$
SRAM Write Cycle Time	PE Cycle Time	$9.52ns$	$5.71ns$
SRAM read Cycle Time	PE Cycle Time	$12.45ns$	$6.74ns$
Shift Operation Cycle Time	Cycle Time	$17.96ns$	$11.48ns$
ALU Operation Cycle Time	Cycle Time	$6.08ns$	$3.9ns$

summarized in Table 4.4. The maximum speed of operation, based on the ALU cycle, is 164MHz, assuming that signals can be supplied precisely. The maximum operational speed for accessing memory externally is 92MHz or 92Mb/s, while reading is 103MHz. The maximum operational speed for writing memory internally for the PEs is 105MHz; or 13.4Gb/s bandwidth for all 128 PEs. Reading memory internally for the PEs results in a maximum operational speed of 80MHz.

In Table 4.4, when the operating voltage is changed from 1.5V to 1.2V, some of

Table 4.4: Maximum Operational Frequencies for 1.2V & 1.5V

Circuit	$1.2V$	$1.5V$
ALU Operation	164MHz	256MHz
Shift Operation	55MHz	87MHz
External Memory Write Operation	92MHz	121MHz
External Memory Read Operation	103MHz	129MHz
Internal Memory Write Operation	105MHz	175MHz
Internal Memory Read Operation	80MHz	148MHz

the circuits undergo a remarkable change in the maximum operational frequencies. The circuits that seem to undergo major changes are the ALU operation and the shift operation, at about 64% of the 1.5V frequency. Looking at the the voltages, it can be seen that 1.2V is 80% of 1.5V. From this it may be expected that the maximum frequency for 1.2V should be about 80% of the maximum frequency for 1.5V operation, and this is seen in the external memory operations.

To ensure that off-chip data shifting completes in the cycle time of the shift operation, the pins and external shift interface must run faster than 87MHz for 1.5V operation. For an 8x8x8 cube having a 8-bit shift interface, the shift pins and the external shift interface must be able to transfer data at 700MHz for 1.5V operation and 440MHz at 1.2V operation.

4.8 Design Results

The following results are estimates of the power and area of the design made through simulation and layout. This section is included to indicate the power consumption that could be expected if this design was taken to the final stage of fabrication. The areas are estimated based on individual transistor layout of the core circuits of the design. It is included to indicate the overhead that would be introduced by a fault-tolerant dynamically reconfigurable C•RAM communication network.

4.8.1 Power - Estimation

The focus of the design of the fault-tolerant, dynamically reconfigurable C•RAM communication network was not on designing for low-power. It is, however, of interest to know the power consumption of such a design. It is possible to determine the power consumed per cycle of the design, and consequently possible to determine if the design is low-power. The design is implemented using the SOI technology, which has the ability to operate at higher frequencies at lower voltages than conventional bulk CMOS technologies of the same voltage level. This design, however, is not implemented in a bulk CMOS technology, due to the limiting factor of creating a 3D stack of chips. It could have been possible to implement the same die design in a bulk CMOS technology to determine single die comparisons, but this was not done.

In this section, the power requirements of the core circuits are measured. For some of the circuits, the power of a single circuit is measured and then scaled to indicate the power used by the total circuit. For example, one sense amplifier is measured for its dynamic power usage, and then the measured value is scaled by the number of active sense amplifiers during the cycle. The dynamic power is also scaled using the activity factor (α), since the probability of all sense amplifiers changing from 0 to 1 is not necessarily 100%. The dynamic power is measured or estimated, as it will give the maximum power draw for the circuit during operation. As most of the circuits are designed as complementary CMOS gates, the static power consumption should ideally be nearly zero. We know, however, this is not completely true due to the sub-threshold leakage of the NMOS and PMOS transistors in MIT LL's SOI technology.

In Table 4.5 and Table 4.6, the dynamic power calculations for the PE and memory array are calculated using the dynamic power formula:

$$P_{DYN} = \alpha C_{LOAD} V_{DD}^2 f$$

The tables contain the circuits that are localized to the array so that the dynamic power for the array is calculated rather then measured, due to the problems of simulating the complete array. When simulating the array, the Cadence multiplier parameter is used to estimate the timing needed for proper functionality. The current measurement gained through simulation using the multiplier parameter will only be an upper estimation, as almost all of the array circuits will be performing exactly the same operation. For this reason, it was decided that the dynamic power use for the array would be calculated using the dynamic power formula. All calculations are for a 4x8x4 PE array with redundant elements for the x and y dimensions of the array.

Table 4.5: Dynamic Power Calculations for 1.5V Operation

Circuit	Quantity	Operating Period (ns)	Load (fF)	α	Dynamic Power (mW)
PE Row Drivers	3	6.74	1755	1	1.76
PE Signal Drivers	16	6.74	827	1	4.41
PE Bit-lines	1200	6.74	774	0.5	155
PE SRAM Cells	600	6.74	5.8	0.5	0.58
PE Sense Amplifiers	600	6.74	5.8	0.5	0.58
IO Row Drivers	1	8.20	1755	1	0.48
IO Signal Drivers	20	8.20	827	1	4.54
IO Bit-lines	400	8.20	774	0.5	42.4
IO SRAM Cells	200	8.20	5.8	0.5	0.16
IO Sense Amplifiers	200	8.20	5.8	0.5	0.16
Control Drivers	32	3.90	774	0.5	7.14
ALU Opcode Drivers	32	3.90	774	0.5	7.14
ALU Result Latch	200	3.90	12	0.5	0.69
Registers (X,Y,WE,S)	800	3.90	6	0.5	1.38
Total					226.42

In Table 4.5 and Table 4.6, the power requirements for the 1.2V operation is less than the 1.5V operation. This is due to the decrease in voltage, but most of this decrease is due to the reduction in the maximum operating frequency of the circuits. This suggests that if the main requirement is low-power, the circuit should be run at 1.2V rather than 1.5V. As already determined, the chip would run at about 100MHz for a 1.2V supply.

Table 4.6: Dynamic Power Calculations for 1.2V Operation

Circuit	Quantity	Operating Period (ns)	Load (fF)	α	Dynamic Power (mW)
PE Row Drivers	3	12.5	1755	1	0.61
PE Signal Drivers	16	12.5	827	1	1.52
PE Bit-lines	1200	12.5	774	0.5	53.4
PE SRAM Cells	600	12.5	5.8	0.5	0.20
PE Sense Amplifiers	600	12.5	5.8	0.5	0.20
IO Row Drivers	1	10.8	1755	1	0.23
IO Signal Drivers	20	10.8	827	1	2.21
IO Bit-lines	400	10.8	774	0.5	20.6
IO SRAM Cells	200	10.8	5.8	0.5	0.08
IO Sense Amplifiers	200	10.8	5.8	0.5	0.08
Control Drivers	32	6.08	774	0.5	2.94
ALU Opcode Drivers	32	6.08	774	0.5	2.94
ALU Result Latch	200	6.08	12	0.5	0.28
Registers (X,Y,WE,S)	800	6.08	6	0.5	0.57
Total					85.86

In Table 4.7, the power for periphery circuits are shown. These power values were obtained through simulation rather than calculation, since the periphery is not regularly structured. From the table, the periphery results in large peaks in power requirement. These peaks are a result of a large number of gates that are dependent on the switching of other gates, such that a large number of gates are switching simultaneously. In the memory core a large amount of the switching is being performed by the signal drivers. These signal drivers are driving the largest capacitive load with minor result changes occurring within each PE or sense amplifier. The peaks in the periphery, however, are averaged with the other switching into the average power, and thus, the average power of the periphery circuits is smaller. If the design is scaled to more than one die and more than one array block per die, only the power of the row decoder and shift circuitry would need to be scaled to get the total power of the design.

Table 4.7: Dynamic Power for Select Circuits

Circuit	1.2V Operation		1.5V Operation	
	Peak Power (mW)	Avg. Power (mW)	Peak Power (mW)	Avg. Power (mW)
Row Decoders	30.2	9.8	79.5	25.8
Periphery	406.6	38.2	856.5	55.2
Shift Circuitry	15.4	1.14	40.5	3
Total	452.2	49.14	976.5	84

4.8.2 Area - Estimation

During the design process, it was of interest to discover the approximate area that would be used by the redundancy and shift circuitry. During this time, attempts were made to derive the amount of space used below the PEs for the wiring of the shift and redundancy circuits. The following calculations assume that the three-dimensional cube on a single die is split on the y-dimension, such that half of the PEs are placed above and the other half below the memory array. The split is performed on the y-dimension because it seemed the most logical way to split the cube. See Figure 4.31 to observe how the PE layout might look as a 2D grid split on the y-dimension. This layout can be easily extended to a 3D cube. Where the lines are PE interconnection wires.

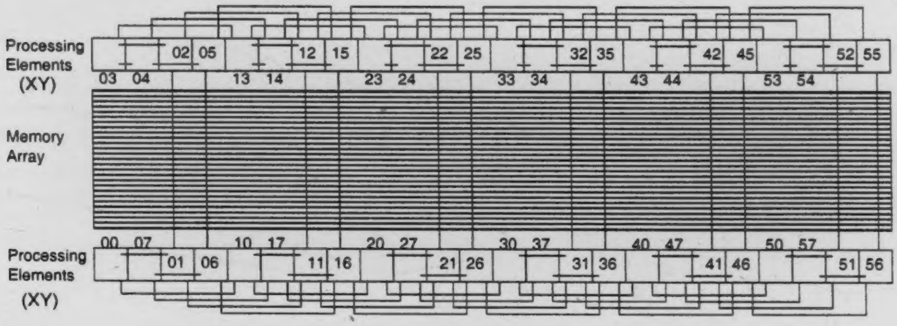

Processing Elements (XY)
02 05
12 15
22 25
32 35
42 45
52 55
03 04
13 14
23 24
33 34
43 44
53 54
Memory Array
00 07
10 17
20 27
30 37
40 47
50 57
Processing Elements (XY)
01 06
11 16
21 26
31 36
41 46
51 56

Figure 4.31: Possible PE placement for 2D Grid

The following formulas are specific to this design and the method of layout used. For the redundancy circuits, the amount of space used explicitly for wiring can be obtained from the following formula:

$$h_{red\_wire} = \frac{y_{dim}/2 \times z_{dim}}{N_{horz\_met\_layers}} \times num_{red\_signals} \times (wireSpc + wireW)$$

For the shift circuits, the amount of space used explicitly for wiring can be obtained from the following formula:

$$h_{shift\_wire} = \left(\frac{y_{dim} z_{dim} + y_{dim}}{2} + \frac{x_{dim} + y_{dim}/2 + z_{dim} - N^2_{horz\_met\_layers}}{N_{horz\_met\_layers}} \right) \times (wireSpc + wireW)$$

For the formulas above, the size of the cube in the x, y and z dimensions are x_{dim}, y_{dim} and z_{dim}, respectively. The spacing between the signal wires is designated by wireSpc, while the width of the signal wires is designated by wireW. The number of signals used for redundancy is designated by $num_{red\_signals}$, and the number of horizontal metal layers available for routing signals horizontally is designated by $N_{horz\_met\_layers}$. This assumes that the PEs are placed horizontally within the PE array.

In Table 4.8, sample calculations are performed to show the impact that the dimensions of the cube have on the routing height for the redundancy and shift circuits. In the table, it is also possible to see how the number of available horizontal metal layers affects the routing height. From this table it would seem reasonable to use an 8x8x4 (x,y,z) cube instead of a 4x8x4 cube since there is almost no additional area penalty in doing so. In Table 4.9, however, the length of the wires for routing of the redundancy, shift, and control signals show that the control signal wires will increase in length. This increase in length adds additional capacitance and resistance to the wire, which means that the delay of the signals increases. This increase in delay reduces the maximum frequency of operation of the design. In addition, by moving from the 4x8x4 cube to the 8x8x4 cube, there is a doubling in the number of processors directly connected to the same control signals.

One of the advantages of this design is that two blocks of 4x8x4 PEs can be connected together to form an 8x8x4 cube of PEs without adversely affecting the delay or maximum speed of operation.

In Table 4.10, area approximations are given for a 4x8x4 PE array that includes redundancy. The table shows the area taken up by the SRAM array, the area taken up by the addition of C•RAM PEs, and the area of the periphery. It should be

Table 4.8: Height Approximations of Shift and Redundancy Wiring

Cube Dimensions (x,y,z)	# Horz Metal Layers	Metal Width μm	Metal Spacing μm	Redundant Signals	Redundancy Height μm	Shift Height μm
4x8x4	2	0.25	0.5	7	43.4	18.6
8x8x4	2	0.25	0.5	7	43.4	20.15
8x8x8	2	0.25	0.5	7	86.8	34.1
8x8x8	1	0.25	0.5	7	173.6	43.4
8x8x8	3	0.25	0.5	7	57.9	18.35

Table 4.9: Length Approximations of Shift and Redundancy Wiring (single die)

Cube Dimensions (x,y,z)	SRAM Height μm	PE Width μm	Redundancy Wire Length μm	Shift Wire Length μm	Control Signal Wire Length μm
4x8x4	6.15	33.55	671	1678	3355
8x8x4	6.15	33.55	671	1678	6039
8x8x8	6.15	33.55	1342	3355	12078

noted that by replicating the 4x8x4 PE array, the area for the periphery remains unchanged. Periphery is not included in the base area or the overhead, as it is the combined total for both, and is unaffected by the replication of the base 4x8x4 array.

From the table, it is seen that the total overhead added by C•RAM is 12.5%, and that the reconfigurable interconnection network adds only 5% of the overhead to the total area. Note that the addition of redundancy adds more than 5%, because the addition of redundancy requires additional SRAM columns that may not be accessible.

In Table 4.10, Boundary Cct is the boundary interconnection circuit used for creating the spiral and torus networks for 1D, 2D and 3D interconnection. The Interconnect Cct is the interconnection circuits for connecting PEs within the planes and connecting the planes; it does not include the boundary interconnection. The redundancy circuits area of the table is only for performing grid redundancy for each plane of PEs.

In Table 4.10, it should be noted that two row decoders are used in calculating the total area. This is for the case of using only two addresses and two row decoders

Table 4.10: Area Approximations for 4x8x4 Array with Redundancy

Circuit	Dimensions		Area	Qty.	Total	Area
	x(μm)	y(μm)	μm^2		mm^2	Percentage
Memory Cells	33.55	6.15	206	25600	5.3	66.25%
Signal Driver Ccts	150	2.5	375	1228	0.5	6.25%
Sensing & Precharge	33.55	67.9	2278	200	0.5	6.25%
Row Decoder	35.4	1574.4	55734	2	0.1	1.25%
Column Decoder	3355	35.4	118767	2	0.2	2.5%
Base Area (SRAM)				1	6.6	82.5%
Processing Element	33.55	25.125	843	200	0.2	2.6%
Redundancy Circuits	3355	65	218075	2	0.4	5.0%
Interconnect Circuit	3355	32.5	109038	2	0.2	2.5%
Boundary Circuit	3355	60.5	202977	1	0.2	2.5%
Overhead (C•RAM)				1	1.0	12.5%
Total Periphery	300	1400	420000	1	0.4	5.0%
Total Area				1	8.0	100%

for the multi-bank design. In this way the two row decoders were built on a single side of the memory array. This was done to ensure that the design could be extended to four decoders, if the specifications required it. If the design was changed to include four row decoders the total area would increase by an additional $0.1mm^2$.

The table shows the area of four banks of memory per PE; however, a PE is pitch matched to eight banks of memory. The area of the SRAM cell is $24.6\mu m^2$ for a multi-bank containing four banks. For a one and two banks of memory per PE, the area of the SRAM cell is $22\mu m^2$. The width of the SRAM cell remains constant at $4\mu m$, while the height of the SRAM cell increases or decreases as banks are added or removed. The height of the SRAM changes, since the word-lines for each bank are routed through every SRAM cell.

It should also be noted that the total area does not include the input and output pad frame. This design is more than likely to be pad limited in size then it is to be limited by the core area of the design.

In Figure 4.32, a bar graph of the areas in Table 4.10 is shown. This figure gives a quick overview of how the area of the core of the die is divided.

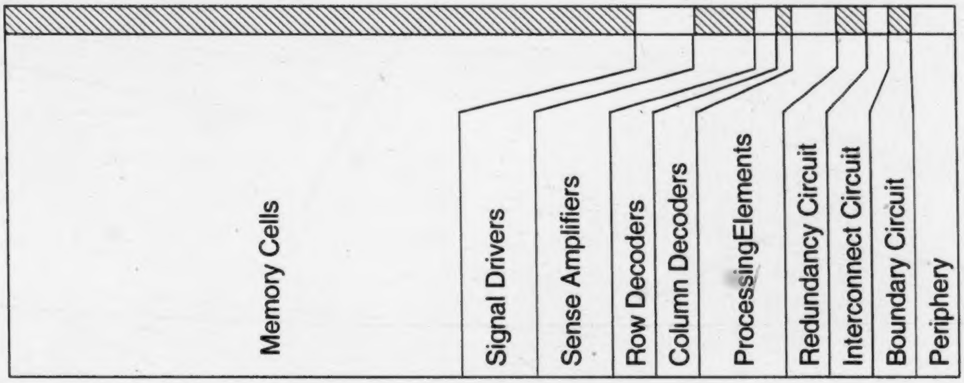


Figure 4.32: Area Bar Graph

Table 4.11: Area Approximations for 4x8x4 Array Redundancy

Circuit	Dimensions		Area	Qty.	Total	Percentage
	x(μm)	y(μm)	μm^2		mm^2	
Memory Cells	33.55	6.15	206	9216	1.90	23.75%
Sensing & Precharge	33.55	67.9	2278	72	0.16	2%
Processing Element	33.55	25.125	843	72	0.01	0.13%
Redundancy Circuits	3355	65	218075	2	0.4	5.0%
Boundary Cct.	3355	60.5	202977	1	0.2	2.5%
Redundancy Area					2.67	33.38%

As was discussed above, it was of interest to determine the amount of area overhead that was added by the redundancy and interconnection network. In Table 4.10, we see that the interconnection network adds 5.0% of overhead. This area may be more when all measures of redundancy are removed from the design. In Table 4.11, the amount of area taken up by redundant memory cells, processing elements, and redundancy mechanisms is about 33% of the total area of a 4x8x4 array containing redundancy. It should be noted that if the size of the array were scaled, the amount of overhead added by redundancy would be reduced. A 4x8x4 array containing redundancy has 72 redundant PEs, meaning 36% of the PEs are redundant, as opposed to a 16x16x4 array with 132 redundant PEs corresponding to 11% of the PEs being redundant.

4.8.3 Yield

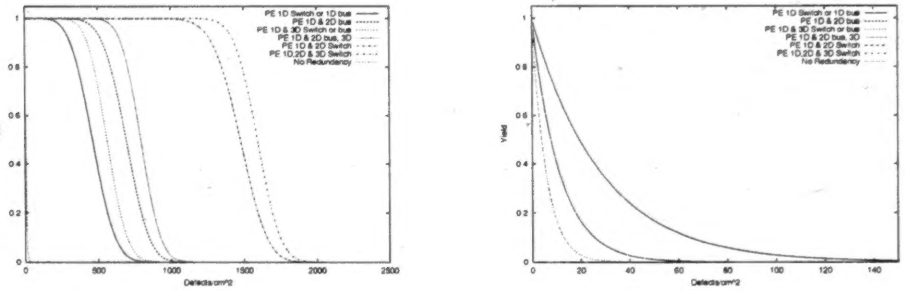
As was discussed in the previous chapter, the cost of yield improvements will dictate whether the yield improvement is worth the cost of the additional area. The goal of designing a chip package is to obtain the highest possible die yield, and to minimize the cost of the package. In the following sections, the yield versus cost for the redundancy designs proposed and developed in this thesis are shown and discussed. For the plots, an 8x8x8 fault-free cube is assumed that contains 256 bits per PE with 4 redundant rows.

For yield, global defects, such as mask misalignment or incorrect implant levels, are ignored as they are not repairable by redundancy. The Poisson model is used for calculating the yield for this design. This model takes into account defect density and the area of a critical region, such as a PE. Fault clustering is not modeled. This model; however, tends to give pessimistic yield estimates. For yield modeling, the die is divided into the three parts that are susceptible to defects. Defects can cause faults in the memory cell, processing element (PE) or the periphery. In this analysis, each parts' area is consider to be critical, such that any defect that intersects the area of a memory cell, a PE or the periphery will cause a fault. In Appendix A and Appendix B, the equations for calculating the yield are shown.

4.8.3.1 Yield for PE Redundancy

Previous chapters have discussed that adding redundancy will increase the die yield of a design. Section 3.3.2.3, showed that each level of redundancy improved the response of the die to defects. It was also mentioned that the area cost of the added redundancy must be addressed before the final method of redundancy is selected. In this section, the yield of the die is shown relative to the defect density. In Figure 4.33(a), the yield of the array is shown relative to the defect density. This shows how the defect density will affect the yield for each method of redundancy. The defect density for MIT LL's SOI process is unknown; therefore, if we assume that the defect density is less than 500 $defects/cm^2$, any of the methods of redundancy will result in the array of the die having nearly 100% yield. Unfortunately, this is not the entire result, as there exists circuitry in the die that does not contain any redundancy. The control circuit, the shift interconnect circuit and the redundancy switches are circuits that contain no redundancy. If a defect intersects any of these non-redundant circuits, the die will fail.

In Figure 4.33(b), the redundancy methods are grouped into curves that contain the same non-redundant circuits. This figure shows that implementing the 2D redundant switch or bus grid will actually reduce the overall yield, since it contains more non-redundant circuit area than other redundancy methods. Therefore, the choice as to which method of redundancy to use should be determined by the area cost of each method of redundancy on the yield.



(a) Yield of PE Array

(b) Yield of Die

Figure 4.33: Yield Plots for PE redundancy

In Figure 4.34, the equivalent yield of the PE array is shown. This graph shows how the yield of the redundancy mechanism is related to the cost of the area of the redundancy, relative to an array containing no redundancy. In the graph, the 1D and 2D switch redundancy introduces about 35% of area overhead, which is similar to the bus redundancy mechanisms. The switch redundancy, however, has a better immunity to defects than the bus redundancy methods for the same area. Additionally the 1D bus or switch redundancy only introduces about 12% overhead over the non-redundant array. Therefore the 1D redundancy would hold an 88% yield for up to about 350 $defects/cm^2$, relative to a non-redundant array reaching 88% yield at 1.4 $defects/cm^2$. The Intel process has a defect density of 0.2 to 0.3

$defects/cm^2$ for the Pentium4 in the 130nm process [15], and this process has a lot more research energy involved than the MIT LL SOI process. Assuming that MIT LL's SOI process has a defect density of 5 $defects/cm^2$, the expected yield of the process without redundancy is approximately 50%.

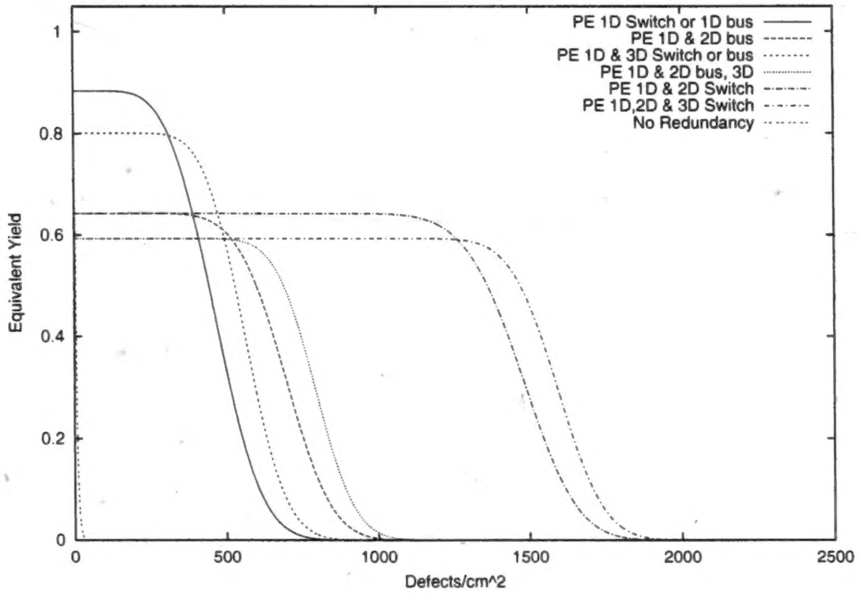


Figure 4.34: Equivalent Yield for PE Array

In further analysis, the array was expanded to have each plane contain a 64x64 fault-free grid (4096 PEs). Accelerix created a C•RAM design that contained 4096 PEs within a single die [1], and it is of interest to stack multiple dies each containing 4096 PEs and determine its affect on the yield. In Figure 4.35, the equivalent yield of a die containing a 64x64 grid of PEs is shown. Each die actually contains a 65x65 grid of PEs, such that 129 PEs are redundant (3%) compared to 20% of the PEs being redundant in a 8x8 (actual 9x9) grid. Knowing that 3% of the area comes from the redundant PEs. Expanding these calculations to three-dimensional redundant meshes; in a 8x8x8 mesh, 30% of the PEs are redundant and in a 64x64x8 mesh, 14% of the PEs are redundant. In both these cases the additional plane adds most of the redundant PEs. From Figure 4.35, we see that for a 64x64 grid, containing 2D redundancy, approximately 15% of the added area comes from the redundancy switches.

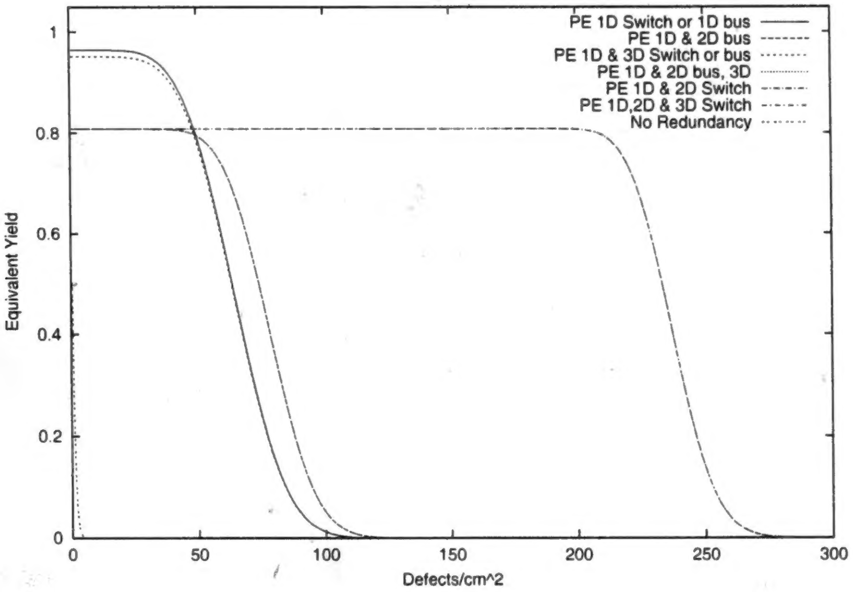


Figure 4.35: Equivalent Yield for 64x64 Fault-tolerant Grid

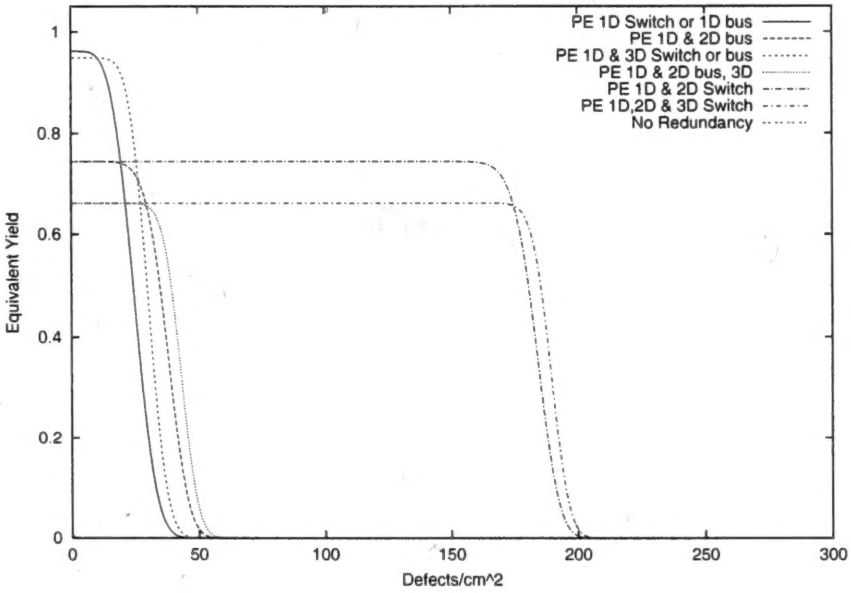
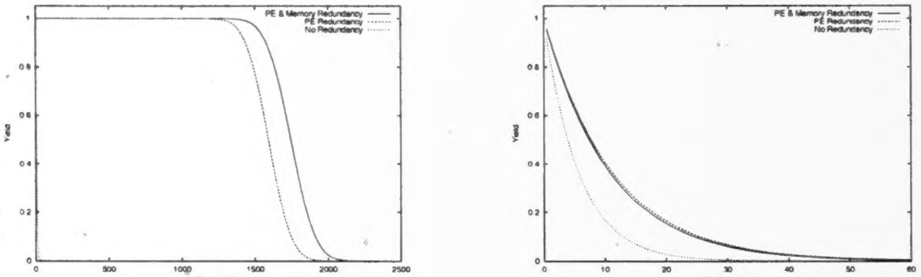


Figure 4.36: Equivalent Yield for 64x64x8 Fault-tolerant Array

In Figure 4.36, the dies are stacked and the results of the redundancy can be seen. The equivalent yield decreases for the addition of 3D redundancy, since the area being used to perform redundancy increases due to the additional plane. Also, the 50% yield is reduced from approximately 240 $defects/cm^2$ to less than 200 $defects/cm^2$ due to the stacking of the dies. In Figure 4.34, the plot assumed that an 8x8x8 mesh was constructed on a single dies instead of separated into a number of dies. This is the reason for a higher equivalent yield for the 64x64x8 mesh, where the mesh was composed of separate dies. This shows that dividing the mesh into multiple dies reduces the area impact of the routing of the redundancy and interconnect networks.

4.8.3.2 Yield for Memory Redundancy

In this section, the graphs are of a PE array containing all three dimensions of redundancy. This section determines whether the introduction of independent memory and PE redundancy produces enough improvement in yield to offset the area of separating the memory from the PE redundancy. In Figure 4.37(a), the yield improvement is introduced to the array by implementing a redundancy switch mechanism between the memory and the PE. PE and memory redundancy corresponds to the separation of the memory and the PE by a redundancy switch, while PE redundancy corresponds to the memory redundancy handled by the PE redundancy. In



(a) Yield of Array

(b) Yield of Die

Figure 4.37: Yield Plots for Memory Redundancy

Figure 4.37(b), the additional area added by the redundancy switches between the memory and the PE actually reduces the overall yield of the die.

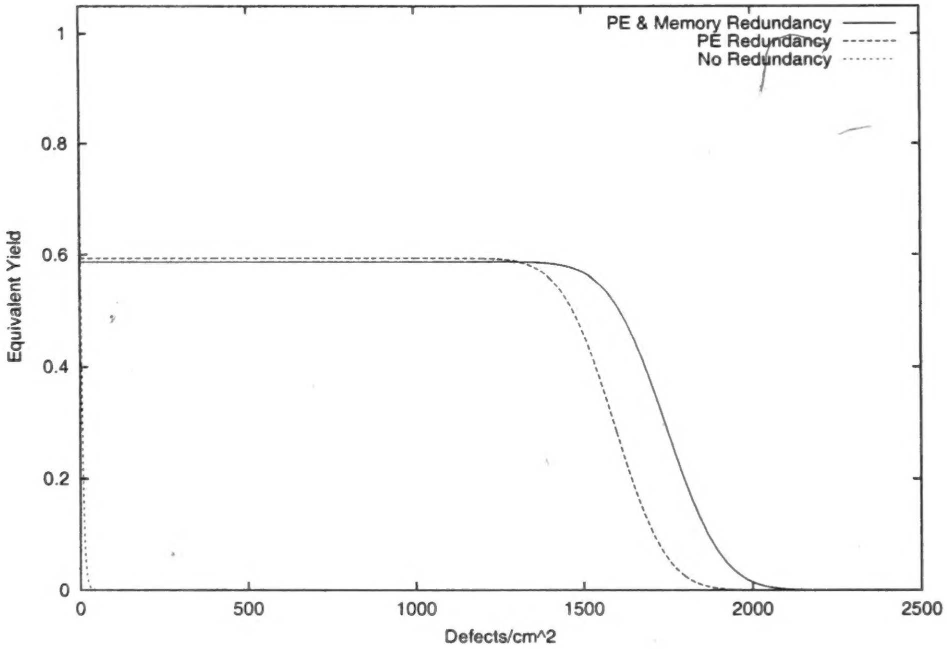


Figure 4.38: Equivalent Yield for Memory Redundancy

In Figure 4.38, the area cost on the yield is small, and the yield improvement is also small. This indicates that the memory redundancy mechanism of a redundancy switch between the memory and PE is not worth the additional cost, as it does not provide enough of an advantage to offset the area and yield cost.

4.9 Summary

In this chapter, the modifications defined in Chapter 3 have been applied to the C•RAM architecture, showing that a dynamically reconfigurable and redundant PE array could be designed and simulated. Simulation of the design resulted in the determination of the power usage, speed of the design, and ultimately in an estimation of the area. It was noted that the design could be run at both 1.5V and 1.2V, with the maximum speed and power usage of the design being reduced when operating at 1.2V. It was also determined that the yield cost of separating the memory and PE for redundancy did not have enough benefit to warrant it being included in a fabrication of this design. The yield cost of the 1D and 2D switch redundancy was identical to that of a 1D and 2D bus redundancy mechanism, with the switch redundancy having a greater immunity to the defect density of a process. It was also discovered that the yield cost of adding 3D redundancy to the switch redundancy was quite high for the limited improvement in overall yield. If 3D redundancy is to be added, it should not be added at the die level, but rather at package level. To add redundancy at the package level, additional wafers should be added to the stack. It is not possible to add good dies to a stack, since it is difficult to line up single dies. The wafers are stacked untested in the MIT LL 3D SOI process.

Chapter 5

Testing

To ensure that the design will operate properly in silicon, it is necessary to ensure that the technology to be used is viable. Fortunately, a C•RAM design has already implemented in this process, which has been called 3DSOI C•RAM [16]. Wafers for the previous SOI C•RAM design have been manufactured. The 3D stacking of the wafers is still to be completed. The manufactured wafer was diced, bonded in an IC package and tested. These chips were sufficiently functional to allow for the running of test programs. The testing of the 3DSOI C•RAM design characterizes the SOI process as it pertains to C•RAM. This will allow the extraction of some useful information that may aid in further characterization of the design performed in this paper.

5.1 Testing of First Generation 3DSOI C•RAM

In order to prove the 3DSOI C•RAM design, it is necessary that chip be able to run a fully parallel program with little or no need for communication with an external host controller. To perform this, the C•RAM compiler program developed by Duncan Elliott [8] was modified to export test vectors that could be used on an HP81200 VXI Tester, as well as an University of Alberta designed FPGA tester [19]. Modifying the compiler to export test vectors specific to the current 3DSOI C•RAM chip made it possible to program any C•RAM based operation, and export properly sequenced test vectors for each operation to the chip. In modifying the compiler, the chip can be tested under several programming conditions to check for chip functionality.

While at the University of Alberta, I developed a program that allows the user to create a custom GUI program that can run the HP81200 VXI Tester and control multiple power supplies simultaneously. In this way, the limitations of the HP81200 Software were removed to allow for fully customizable tests that made use of power supply controlling to be run. This has allowed for the creation of programs that run on the HP81200 VXI Tester using HP power supplies connected using the GPIB interface. A program that varies the power supplies and the frequency of operation was built that allowed for the creation of shmoo plots. Shmoo plots, typically plot the pass and fail of a device relative to frequency of operation, and voltage, and sometimes temperature. Therefore, to create the shmoo plots a program that obtains the results of the chip device under multiple frequencies and voltages was designed, called shmoo program. The shmoo program uses input files containing test vectors and shmoo environment data to setup the tester, the program can be executed and requires no intermediate interaction until the results are ready. With the HP software, users are required to sit at the workstation, continually changing the test-vector file and power-supplies, this is very time consuming.

5.1.1 Proof of 3DSOI Technology

The 3DSOI C•RAM design was created to allow for die stacking, as well as single die execution. At the time of writing and testing, the single die packages were the only packages available. It still allows testing of the functionality of the chip. There are also a number of circuits in the design that are used to identify and enable each die in the stack; these circuits can still be tested for rough functionality, since if they fail, the chip will be un-addressable. The outputs of the chip will remain unchanged, since all outputs depend on the die identification circuits.

The 3DSOI C•RAM design was implemented in a FDSOI transistor technology. As described in earlier chapters, there are some advantages and disadvantages to using a Fully-Depleted SOI technology. The FDSOI has better operational ability in the absence of a body contact in comparison with a PDSOI technology. The 3DSOI C•RAM was simulated using spice models provided by MIT, and through simulation, the design was optimized to provide a PE operational speed near 500MHz. The testing of the 3DSOI C•RAM will give a good indication of how closely the actual design matches the simulated results.

Unfortunately, no equipment was available to test the C•RAM chip near their estimated 500MHz maximum operational speed. The maximum testing speed available at the University of Alberta was 200MHz using the Agilent HP81200 tester. The actual maximum testing speed that was possible was 33MHz, since the prototype board is built using wire-wrap and testing above this speed resulted in unexplainable errors. The purpose, however, is only to prove the functionality of the design. The design should be extremely reliable at 33MHz, and therefore it should be possible to determine the functionality of the design.

5.1.1.1 3D Technology - Die Extended Signals

For C•RAM, it has been mentioned that the design was intended to allow for stacking of multiple identical dies in an attempt to create a device that is scalable within a package. To do this, additional circuitry was needed that allowed for signals to propagate through the multiple dies of the package while maintaining the signal properties. For example, a shifting bus must maintain the left and right interconnect linearity between the dies and the external interface. Therefore, the signal must propagate through the other dies if it is not to be used on that die.

Testing of the C•RAM device began by testing the databus and die addressing. The die addressing is essential for all memory accessing, and therefore the databus. If the die addressing does not work, then writing to the databus will have no effect on the databus output. To test the databus, the die address was left at 0, indicating that an attempt to access the first die in the stack. After ensuring that the die address is properly set, the databus is written to and the output is analyzed. To ensure proper databus operation it is necessary to write all possible patterns to the databus. To test the die addressing, the die address is changed, indicating a different die in the stack.

Since only one die is available, the databus manipulation will result in no change on the data output pin when the die address is not 0. During testing, we determined that the databus and the die addressing were fully functional.

To begin testing the 3D technology aspect of the shift network, the PEs on the ends of the array were toggled to ensure that the shift out pins also toggled. Performing this operation, we determined that the die interconnect was properly configured, so as not to interfere with individual die packaging.

A similar test was performed for the broadcast bus, and results indicated that the broadcast bus was still operational for single die packaging.

These four simple tests helped to prove that the 3D circuits that were developed operate properly in the absence of a three-dimensional stack.

5.1.2 Test Strategy

The testing of 3DSOI C•RAM can best be performed by testing the PEs of the chip first. Once the PEs are tested, they can be used to test the memory array in parallel. This is possible because of the parallel nature of C•RAM, which implies that each PE can be used to test the column of memory that it is associated with. Once all the PEs that have failed are determined, it is possible to test all the memory above the PEs by writing to them manually through the databus. In this way, the yield of the memory array apart from the PEs is determined, as well as the yield of the PEs apart from the yield of the memory array. The best redundancy scheme for compensating for failures in both the memory array and the PEs can then be determined.

For 3DSOI C•RAM , all the control signals are supplied externally. This allows us to change the order in which control signals are applied, as well as the time between control signals. In this way, we have full control over the chip and its operation. We have full control over the instructions being supplied to the PEs, the enabling of the broadcast bus, the enabling of the shift network, and reading and writing to the registers of the PE. We also have full control over the memory array above the PEs. The full control over 3DSOI C•RAM was built because it was to be a test of the C•RAM architecture in a SOI process.

To test the PEs of C•RAM, it is best to run through all of the PE operations that are available. Since all PEs share the common broadcast bus, it is possible to use the broadcast bus to determine faults in the PEs. If the broadcast bus is found to be faulty regardless of the PE state, it can be said that the chip is non-functional. It was determined after the chip was sent for fabrication that the broadcast bus circuit may not have been designed properly and may be non-functional. Therefore, it may be necessary to use other methods to determine PE faults.

5.1.2.1 Testing with Non-Functional Broadcast Bus

When the broadcast bus is non-functional, another method of determining the results of the PE is required. One method is to use the databus so that the results are written to the sense amplifier of each PE, and then read for each PE. This requires, however, that all pins of the databus are fully functional to allow for correct reading of the PE's results. If the databus is not functional, there still exists the possibility of using the shift circuit. By using the shift circuit, we require that every point in the shift circuit must work properly. Yet by using the shift circuit, it is possible to find the first PE with a failure in the shift circuit or in the actual PE. Using the shift method, it would be possible to find at least two failures, since it is possible to shift left and right. Therefore, two shift failures or PE failures can be diagnosed.

This method of testing is very time consuming, since each test must shift out the results every time. It is highly dependent on the shift circuit being fully functional.

5.1.2.2 Testing with Functional Broadcast Bus

At the University of Alberta, X. Sun et al. devised a testing strategy that could be used to determine the functionality of a C•RAM device [21]. This testing strategy relies heavily on the broadcast bus of the C•RAM device being fully functional. In [21], X. Sun proposes that the PEs be tested first, and that it be done independent of the local memory. Once testing of the PEs has verified the functional PEs, the PEs can be used to generate and apply test patterns to the local memory. The testing strategy supplied ignores the availability of skip (S) registers that can be used to disconnect faulty PEs from the broadcast bus. The skip register allows devices that have faulty PEs to still be used; however, with a smaller set of available PEs.

During investigation of the testing strategy, it was noted that the functional diagram in [21] differed slightly from the actual implementation of C•RAM that is tested. The general testing strategy, however, can still be applied to the 3DSOI C•RAM chip to help identify the number of PEs that are functional. The testing procedures of X. Sun is modified slightly to accommodate the actual implementation of C•RAM. The following testing procedures are used: (1) Left/RightShift; (2) ANDbusWalk-0; (3) MMUXtest; and (4) OpcodeWalk-0/1. The AMUXtest, XMUX_XStest and YMUX_YStest are not usable for 3DSOI C•RAM. Once testing of the PEs is performed, the local memory above the functional PEs is tested using the WMUXtest, CRAM_MATS+ and DatabusTest testing procedures.

This method of testing is very fast, since every test is combined on the broadcast bus. If at least one PE fails, the results of the test are seen immediately. Using the broadcast bus, it is possible to set the skip register such that the shift circuit could determine the PEs that passed and the PEs that failed. This method is very dependent on the broadcast bus being fully functional.

5.1.3 Test Results

As mentioned above, only the single die packages were available for testing at the time this thesis was written. It is for this reason that only the test results for the single die packages are included. In Section 5.1.1.1, the test to determine single die functionality was described, and through this testing, it was possible to determine that all the single die packages passed the single die 3D technology tests. The next stage of testing is determine the functionality of the C•RAM array. Using the test strategy described in the previous section, the following results for single die operation are presented.

5.1.3.1 Single Die

The compiler available for C•RAM was used to code the testing procedures that were used to generate test vectors for the 3DSOI C•RAM chips. The 3DSOI C•RAM chip is tested using the HP81200 VXI tester. Initial testing of the 3DSOI C•RAM proved that the broadcast bus was functional at a 10MHz operational speed.

Initial Test Results

Initial testing was performed with the 3DSOI C•RAM chip operating at 1.5V, and vectors being delivered at 10MHz. The ALU operations, however, require from five to ten test vectors per cycle, resulting in a total operating speed of 1 to 2MHz. Through the testing procedures of [21], it was possible to initially diagnose that either some of the Y registers were unable to write '0's, or that the ALU is unable to test Ybar. It was also found that shifting to the right produced incorrect results; from this it can be determined that the problem may exist in the Y registers and not the ALU. This is due to the fact that the Y register is the target for a shift right operation.

In initial testing, shifting left and writing of the X register produced proper results. ALU operations that rely on the X register alone produced right results; however, operations for the Y register or M register produced incorrect results. The skip register was also tested to determine if setting the skip register produced proper results. For instance, when all PEs' skip registers were set to '1', the broadcast bus remained unchanged; when each PE should was broadcasting a '0' to the broadcast bus; this is proper operation. Despite the fact that the skip registers could be properly set to '1', it was not possible to produce a fault-free PE array because the skip register is set based on the M register results. The problem seems to be an ALU issue, since it is not possible to combine the Y and M register results to produce the appropriate skip register setup.

From initial testing it was found that the number of working PEs for the array cannot be determined due to the ALU, Y register, and M register problems. During initial testing it was also found that two of the chips had partially working databuses. For one of the chips, the bottom two databus pins were fully functional, while for the other chip, the top two databus pins were fully functional. From this it can be determined that the databus problems are not design related, but are probably related to possible manufacturing defects or power surges. The 3DSOI C•RAM chips were not designed with ESD protection diodes, and are therefore susceptible to static discharges causing problems. The fact that only two of the databus pins work well on each chip means that only half of the chip can be tested for correct operation and that it is not possible to set the skip registers by using the databus. The chip was designed to allow for the skip registers to be written by writing to the M register and then copying the value of the M register into the skip register.

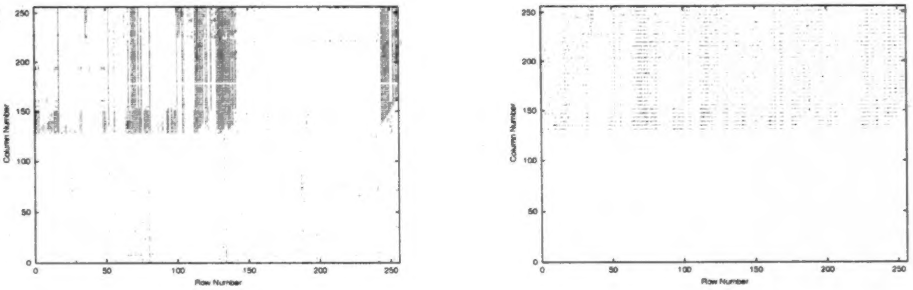
Full Test Results

Full testing of the 3DSOI C•RAM chip involved testing the PEs using the suite of tests developed by X. Sun et al. Since it was found through the test suite that it was not possible to isolate the failed PEs, the testing of the memory array was performed through brute force method. The brute force method of testing was to perform a march test on the memory array, much the same way that X. Sun mentioned for testing using the PEs. The testing of the entire array; however, was performed through the databus using the column addressing rather than PE operations. The test was run with the chip operating at 1.5V and the test vectors being supplied on a 10MHz clock. The test was run on four available packages, with the following results in Table 5.1.

Table 5.1: 3DSOI C•RAM Test Results

Chip No.	Functional Cells	Total Cells	% Good
1	8938	65536	13.6
2	6536	65536	10.0
3	5447	65536	8.3
4	1390	65536	2.1

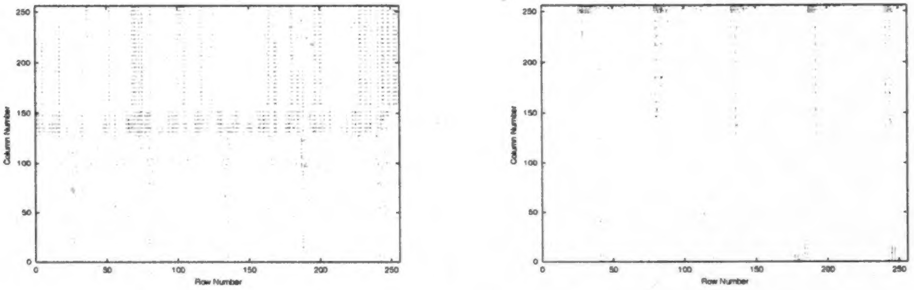
In Figure 5.1 and Figure 5.2, the chip plots of the four chips are shown. Chip 2 seems to be better than chip 1 for half of the chip. From the plots, it can be determined that on chip 2, two bits of the databus are faulty. For 3DSOI C•RAM, the databus is built so that two bits of the databus are used for columns 0 to 127, and the other two bits are used for columns 128 to 255. Since chip 1 seems to have the largest number of working cells, and it appears that all four bits of the databus are working, the shmoo of the device was performed on chip 1.



(a) Chip 1

(b) Chip 2

Figure 5.1: Chip Plots - a black dot represents a good cell



(a) Chip 3

(b) Chip 4

Figure 5.2: Chip Plots

For the shmoo of the device, it was decided that the frequency range used to supply the vectors would be 1MHz to 31MHz, and the operational voltage would range from 1.0V to 1.8V. The shmoo of the device is performed on the memory array, to determine the best operational voltage and frequency for supplying the test vectors. During the shmoo of the device, it was found that only 1445 cells were common to each operating point. For this reason, the shmoo plots for this device are not shown as a pass or fail, but rather as contour plots showing how many cells passed at each operating point. The contour shmoo plot of good cells is seen in Figure 5.3.

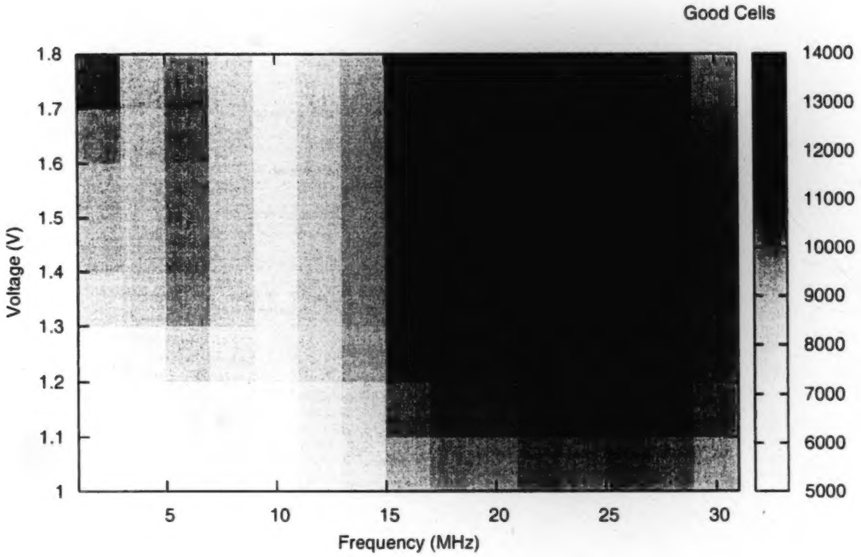
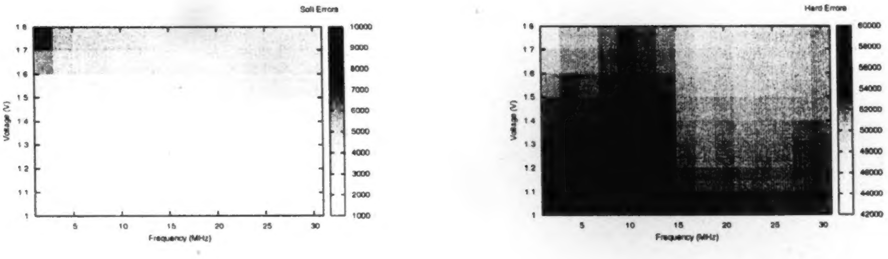


Figure 5.3: Contour Shmoo Plot of Good Cells of Chip 1

Since in testing it is possible that multiple runs may result in slightly different results, chip 1 was run through the shmoo test several times. Figure 5.3 shows the results of all the cells that were good for all the tests. Each test results in cells that have errors, which can be separated into errors that occurred in all tests, and errors that occurred in at least one test. Errors that occur in one or more tests are soft errors; they do not always occur. Errors occurring in all tests are defined as hard errors, since they always occur. In Figure 5.4, the number of soft and hard errors for each operating point are shown.

From the results of the shmoo, we found that operating at less than 10MHz resulted in a lower number of working cells than operating above 15MHz. The result of poorer operation at a lower frequency suggests that some of the circuits in the



(a) Soft Errors

(b) Hard Errors

Figure 5.4: Contour Shmoo Plot of Errors of Chip1

chip have poor responses at lower frequencies. This may be due to the subthreshold leakage or floating body effects of the SOI process, which only become apparent at lower frequencies where the time between cycles is too large. From the plot, it is determined that 3DSOI C•RAM be operated at 1.5V and 17MHz to 19MHz for optimal functionality.

5.2 Summary

From the test results of 3DSOI C•RAM, it has been determined that the use of a pass-transistor ALU may have contributed to the PE failures. It appears that the ALUs do not pass signals for all 256 combinations of the ALU. This makes the operation of the PEs questionable, and the results cannot be completely trusted. In addition, during testing, it was determined that the memory array and databus of 3DSOI C•RAM were less than acceptable. The failures in the memory array may be due to defects in the SOI process. The process was quite new, and the defect density of the the process was unknown at the time of designing. It was also discovered that the simulation parameters for SPICE were found to be lacking, so that the simulation results were under question. It appeared that the NMOS and PMOS transistor models were extremely leaky, resulting in some design choices being forced. It was found that the optimal operating point for 3DSOI C•RAM is 1.5V, with test vectors being supplied on a 17MHz to 19MHz clock frequency.

Chapter 6

Conclusions

This work presents the results from research into implementing a dynamically reconfigurable fault-tolerant interconnection network for the C•RAM architecture. A dynamically reconfigurable interconnection network capable of one-dimensional, two-dimensional and three-dimensional communication was implemented. The design of a dynamically reconfigurable interconnection network required that all methods of redundancy be changed in order to accommodate the reconfigurability of the network, as well as the multi-dimensional nature of the network. In this research, the redundancy methods to accommodate the multi-dimensional nature of the interconnection network were designed and implemented. This work also presented results from research into implementing a memory interface that can perform external loading or storing of data, while the PEs are performing computations using a different address. This was done by attaching multiple banks of memory to each PE, with each bank of memory being individually accessible.

In Chapter 3, the design of a dynamically reconfigurable interconnection network was presented. This interconnection network resulted in the need for new methods of redundancy that would maintain the network despite the failure of multiple processing elements (PEs). The interconnection network was composed of multiple grids of PEs (planes) that could be organized into a cube or a larger grid. The introduction of the cube interconnect was the main reason for the development of a new redundancy method for PE replacement. For the cube to be complete, all the grids (planes) must have a uniform construction to accommodate a patterned connection between the planes.

The design of the multi-bank memory interface was also addressed in Chapter 3. The advantage of allowing two accesses to occur in parallel, accessing data externally with the databus and internally with the PEs, is that more computations and data accesses can be performed per second than was possible using a single bank of memory. This interface increased the PE utilization per second, as well as allowing for more efficient use of the data.

Chapter 4 presented the implementations of the architectural design changes described in Chapter 3. The speed and power usage of the design was given, as well as the area of the design allowed for evaluation of the yield cost for the redundancy designs. From the yield costs, it was found that the addition of redundancy switches between the memory array and the PEs did not produce sufficient improvement to warrant its implementation during fabrication. It was also found that two possible methods of redundancy produced the best yield results for their cost: the implementation of one-dimensional redundancy through either switches or a bus, or two-dimensional redundancy through switches. Three-dimensional redundancy on die level is of little or no advantage. Using the stacking ability of MIT's SOI process, three-dimensional redundancy could be added by stacking additional wafers. The cost of the circuitry to produce three-dimensional redundancy between planes is small as compared to the area added by redundant planes of PEs.

The testing of the prior generation, 3DSOI C•RAMarchitecture is presented in Chapter 5. It was found that chips, for the most part were unusable, since there was not a single chip that could be configured into a complete array of PEs. During testing it was found that less than 14% of the memory cells in the array were functional. We also determined that the pass-transistor ALU may not have been properly sized, or at least, the process does not work well with pass-transistor architectures. It was possible, however, to determine that the broadcast bus was designed properly, as testing revealed correct operation.

In conclusion, the introduction of a multi-bank memory interface and a dynamically reconfigurable multi-dimensional interconnection network improve PE utilization and shifting.

6.1 Future Work

This work demonstrates the improvement in efficiency and PE utilization that can be obtained through modification of the C•RAM architecture. It also demonstrates the need for improvement in the redundancy architecture of C•RAM. The redundancy architecture aimed at producing a fixed size fault-free grid within a faulty grid of PEs is shown. It was also seen that the improvement of yield in the array is directly counteracted by the non-redundant nature of the redundancy switches. Future work should focus on creating a redundancy architecture that is not limited to a fixed size fault-free grid, and on building redundancy into the mechanism used to perform redundancy, which would reduce the impact that a failure in the redundancy mechanism has on the die.

The introduction of another network type, such as Xnet, to the interconnection network could introduce new uses and improve shifting efficiency further.

Bibliography

[1] Noah Aklilu. Integrating Computational RAM (CRAM) into a System Architecture. Master's thesis, University of Alberta, 2001.

[2] Robert J. Baron and Lee Higbie. *Computer Architecture: Case Studies*. Addison Wesley, Reading, Massachusetts, 1992.

[3] J. A. Burns et al. 3D Circuit Integration Technology for Multiproject Fabrication. MIT Lincoln Laboratory, May 2001. DARPA PI Meeting.

[4] J. A. Burns et al. Three-dimensional Integrated Circuits for Low-power, High-bandwidth Systems on a Chip. In *Proc. IEEE Intl. Solid-State Circuits Conf.*, pages 268–269, 453, feb 2001.

[5] C. T. Chuang, P. F. Lu, and C. J. Anderson. SOI for Digital CMOS VLSI: Design considerations and advances. *Proc. of the IEEE*, 86(4):689–720, April 1998.

[6] Sorin Cristoloveanu. *The VLSI Handbook*, chapter 4. CRC Press LLC, 2000.

[7] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie. Computational RAM: Implementing Processors in Memory. *IEEE Design and Test of Computers*, pages 32–41, January-March 1999.

[8] Duncan George Elliott. *Computational RAM: A Memory-SIMD Hybrid*. PhD thesis, University of Toronto, 1998.

[9] Hirohito Kikukawa et al. 0.13-um 32-Mb/64-Mb Embedded DRAM Core with High Efficient Redundancy and Enhanced Testability. *IEEE Journal of Solid-State Circuits*, 37(7):932–940, 2002.

[10] Richard Cliff et al. Implementation of Redundancy on a Programmable Logic Device. Technical Report 5,498,975, US Patent Office, 1996.

[11] Sung Bae Park et al. A 0.25-μm, 600-MHz, 1.5V, Fully Depleted SOI CMOS 64-bit Microprocessor. *IEEE Journal of Solid-State Circuits*, 34(11):1436–1445, 1999.

[12] Terry Fountain. *Processor Arrays: Architecture and Applications*. Academic Press, 1987.

[13] Dr. Kiyoo Itoh. *VLSI Memory Chip Design*. Springer-Verlag, 2001.

[14] Lizy Kurian John and Eugene John. A Dynamically Reconfigurable Interconnect for Array Processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(1):150–157, 1998.

[15] IC Knowledge. *130nm Yields*. Retrieved January 5, 2004, from http://www.icknowledge.com/economics/yields.html.

[16] John C. Koob, Raymond J. Sung, Tyler L. Brandon, Duncan G. Elliott, Bruce F. Cockburn, and Lisa McIlrath. Design of a 3D Fully-Depleted SOI Computational RAM. In *Proceedings of the 28th European Solid-State Circuits Conference*, ESSCIRC 2002, pages 135–138, Firenze, Italy, September 2002.

[17] Israel Koren and Adit D. Singh. Fault Tolearnce in VLSI Circuits. *Computer, Special Issue on Fault Tolerant Systems*, 23:73–83, July 1990.

[18] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St Pierre, David S. Wells, Monica C. Wong-Chan, Shaw-Wen Yang, and Robert Zak. The Network Architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996.

[19] Fang Pang. IC Tester Implemented using ARM Rapid Prototyping System. Meng - ECE, University of Alberta, 2003.

[20] D. Parkinson, D.J. Hunt, and K.S. MacQueen. The AMT DAP 500. In *Digest of Papers: COMPCON Spring 88. Thirty-Third IEEE Computer Society International Conference*, pages 196–199, 1988.

[21] X. Sun, B.F. Cockburn, and D.G. Elliott. An Efficient Functional Test for the Massively-Parallel CRAM Logic-Enhanced Memory Architecture. In *Proceedings of IEEE 18th International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 475–482, Cambridge, MA, USA, November 2003.

Appendix A

Yield Calculations

A.1 Column and PE Bus Redundancy

$$Y_{sc} = e^{-\lambda} \tag{A.1}$$

$$Y_{row} = Y_{sc}^{columns} \tag{A.2}$$

$$Y_{SRAM} = \sum_{i=0}^{rows_{red}} \binom{rows + rows_{red}}{i} Y_{row}^{rows+rows_{red}-i} \left(1 - Y_{row}\right)^{i} \tag{A.3}$$

$$Y_{roweff} = Y_{SRAM}^{\frac{1}{rows}} \tag{A.4}$$

$$Y_{sceff} = Y_{roweff}^{\frac{1}{columns}} \tag{A.5}$$

$$Y_{rc} = Y_{sceff}^{rows} \tag{A.6}$$

$$Y_{PE} = e^{-\rho} \tag{A.7}$$

$$Y_{PErc} = Y_{PE} * Y_{rc} \tag{A.8}$$

$$Y_{group} = \sum_{i=0}^{PE_{xred}} \binom{PE_x + PE_{xred}}{i} Y_{PErc}^{PE_x+PE_{xred}-i} \left(1 - Y_{PErc}\right)^{i} \tag{A.9}$$

$$Y_{0D} = Y_{group}^{\frac{1}{(PE_x+PE_{xred})}} \tag{A.10}$$

$$Y_{1D} = \sum_{i=0}^{PE_{xred}} \binom{PE_x + PE_{xred}}{i} Y_{0D}^{PE_x+PE_{xred}-i} \left(1 - Y_{0D}\right)^{i} \tag{A.11}$$

$$Y_{2D} = Y_{1D}^{PE_y} \tag{A.12}$$

$$Y_{die} = \sum_{i=0}^{PE_{zred}} \binom{PE_z + PE_{zred}}{i} Y_{2D}^{PE_z+PE_{zred}-i} (1 - Y_{2D})^i \tag{A.13}$$

A.2 Dependent Column and PE Redundancy

$$Y_{sc} = e^{-\lambda} \tag{A.14}$$

$$Y_{row} = Y_{sc}^{columns} \tag{A.15}$$

$$Y_{SRAM} = \sum_{i=0}^{rows_{red}} \binom{rows + rows_{red}}{i} Y_{row}^{rows+rows_{red}-i} (1 - Y_{row})^i \tag{A.16}$$

$$Y_{roweff} = Y_{SRAM}^{\frac{1}{rows}} \tag{A.17}$$

$$Y_{sceff} = Y_{roweff}^{\frac{1}{columns}} \tag{A.18}$$

$$Y_{rc} = Y_{sceff}^{rows} \tag{A.19}$$

$$Y_{PE} = e^{-\rho} \tag{A.20}$$

$$Y_{PErc} = Y_{PE} * Y_{rc} \tag{A.21}$$

$$Y_{group} = \sum_{i=0}^{PE_{yred}} \binom{PE_y + PE_{yred}}{i} Y_{PErc}^{PE_y+PE_{yred}-i} (1 - Y_{PErc})^i \tag{A.22}$$

$$Y_{0D} = Y_{group}^{\frac{1}{(PE_y+PE_{yred})}} \tag{A.23}$$

$$Y_{1D} = \sum_{i=0}^{PE_{yred}} \binom{PE_y + PE_{yred}}{i} Y_{0D}^{PE_y+PE_{yred}-i} (1 - Y_{0D})^i \tag{A.24}$$

$$Y_{2D} = \sum_{i=0}^{PE_{xred}} \binom{PE_x + PE_{xred}}{i} Y_{1D}^{PE_x+PE_{xred}-i} (1 - Y_{1D})^i \tag{A.25}$$

$$Y_{die} = \sum_{i=0}^{PE_{zred}} \binom{PE_z + PE_{zred}}{i} Y_{2D}^{PE_z+PE_{zred}-i} (1 - Y_{2D})^i \tag{A.26}$$

A.3 Independent Column and PE Redundancy

$$Y_{sc} = e^{-\lambda} \tag{A.27}$$

$$Y_{row} = Y_{sc}^{columns} \tag{A.28}$$

$$Y_{SRAM} = \sum_{i=0}^{rows_{red}} \binom{rows + rows_{red}}{i} Y_{row}^{rows+rows_{red}-i} \left(1 - Y_{row}\right)^{i} \tag{A.29}$$

$$Y_{roweff} = Y_{SRAM}^{\frac{1}{rows}} \tag{A.30}$$

$$Y_{sceff} = Y_{roweff}^{\frac{1}{columns}} \tag{A.31}$$

$$Y_{rc} = Y_{sceff}^{rows} \tag{A.32}$$

$$Y_{1Dmem} = \sum_{i=0}^{PE_{yred}} \binom{PE_y + PE_{yred}}{i} Y_{rc}^{PE_y+PE_{yred}-i} \left(1 - Y_{rc}\right)^{i} \tag{A.33}$$

$$Y_{PE} = e^{-\rho} \tag{A.34}$$

$$Y_{1DPE} = \sum_{i=0}^{PE_{yred}} \binom{PE_y + PE_{yred}}{i} Y_{PE}^{PE_y+PE_{yred}-i} \left(1 - Y_{PE}\right)^{i} \tag{A.35}$$

$$Y_{group} = Y_{1Dmem} Y_{1DPE} \tag{A.36}$$

$$Y_{0D} = Y_{group}^{\frac{1}{(PE_y+PE_{yred})}} \tag{A.37}$$

$$Y_{1D} = \sum_{i=0}^{PE_{yred}} \binom{PE_y + PE_{yred}}{i} Y_{0D}^{PE_y+PE_{yred}-i} \left(1 - Y_{0D}\right)^{i} \tag{A.38}$$

$$Y_{2D} = \sum_{i=0}^{PE_{xred}} \binom{PE_x + PE_{xred}}{i} Y_{1D}^{PE_x+PE_{xred}-i} \left(1 - Y_{1D}\right)^{i} \tag{A.39}$$

$$Y_{die} = \sum_{i=0}^{PE_{zred}} \binom{PE_z + PE_{zred}}{i} Y_{2D}^{PE_z+PE_{zred}-i} \left(1 - Y_{2D}\right)^{i} \tag{A.40}$$

Appendix B

2D Switch Grid Redundancy Yield

Yield of a single PE, where A_{PEc} is the critical area of the PE and D_0 is the defect density.

$$Y_{PE} = e^{-\lambda} = e^{-A_{PEc}D_0} \tag{B.1}$$

Using the grid redundancy method there exist a number of configurations of four faulty PEs that will result in a plane failure. This equation calculates the number of configurations that exist for four PEs that will kill a plane of PEs.

$$\begin{aligned} num_{4kill} &= x_{PE}y_{PE} + x_{PE}(y_{PE} - 3 + \sum_{i=1}^{y_{PE}-3} i) \\ &\quad + y_{PE}(x_{PE} - 3 + \sum_{j=1}^{x_{PE}-3} j) \\ &\quad + (x_{PE} - 3 + \sum_{j=1}^{x_{PE}-3} j)(y_{PE} - 3 + \sum_{i=1}^{y_{PE}-3} i) \end{aligned} \tag{B.2}$$

Since we know that the grid redundancy mechanism can repair upto four faulty PEs with no problem, the summation is separated. The first summation finds the yield assuming zero to three faulty PEs. For four PEs upto the number of redundant PEs available in the grid, we subtract the number of four PE configurations that kill the plane from the value of the choose operation. This is that one or more four PE combinations may occur that will cause the plane to fail; however, only one of these four PE combinations needs to occur to make the plane fail.

$$\begin{aligned} Y_{2D} &= \sum_{i=0}^{3} \binom{PE_{xy}+PExyred}{i} Y_{PE}^{PE_{xy}+PE_{xyred}-i} (1 - Y_{PE})^i \\ &+ \sum_{i=4}^{PE_{xyred}} \left(\binom{PE_{xy}+PExyred}{i} - num_{4kill} \right) Y_{PE}^{PE_{xy}+PE_{xyred}-i} (1 - Y_{PE})^i \end{aligned} \tag{B.3}$$

In this way the yield of the grid is actually smaller than if the grid could be completely reconfigured to allow the replacement of PEs until all the redundant PEs are used up before failure occurs. Similarly Y_{PE} could be replace by Y_{0D} in Appendix A yield formulas.

The yield of the array also depends on the number of planes built within a die.

$$Y_{array} = \sum_{i=0}^{PE_{zred}} \binom{PE_z + PE_{zred}}{i} Y_{2D}^{PE_z+PE_{zred}-i} \left(1 - Y_{2D}\right)^i \qquad \text{(B.4)}$$

Finally, the die yield also depends on the yield of the periphery circuits and the circuits that do not contain redundancy. These circuits are circuits that can cause the die not to function if the defect occurs in this area. A_{NRc} is the critical area of the non-redundant circuitry, such as the periphery controller, the interconnection network and the redundancy switches.

$$Y_{periphery} = e^{-\rho} = e^{-A_{NRc}D_0} \qquad \text{(B.5)}$$

$$Y_{die} = Y_{array} \times Y_{periphery} \qquad \text{(B.6)}$$